MATRIXx™

Template Programming Language User Guide

April 2004 Edition
Part Number 370864A-01

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599,
Canada (Calgary) 403 274 9391, Canada (Ottawa) 613 233 5949, Canada (Québec) 450 510 3055,
Canada (Toronto) 905 785 0085, Canada (Vancouver) 514 685 7530, China 86 21 6555 7838,
Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 385 0 9 725 725 11,
France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, Greece 30 2 10 42 96 427, India 91 80 51190000,
Israel 972 0 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400,
Malaysia 603 9131 0918, Mexico 001 800 010 0793, Netherlands 31 0 348 433 466,
New Zealand 0800 553 322, Norway 47 0 66 90 76 60, Poland 48 22 3390150, Portugal 351 210 311 210,
Russia 7 095 783 68 51, Singapore 65 6226 5886, Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197,
Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51, Taiwan 886 2 2528 7227,
Thailand 662 992 7519, United Kingdom 44 0 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on the documentation, send email to techpubs@ni.com.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

AutoCode™, DocumentIt™, MATRIXx™, National Instruments™, NI™, ni.com™, SystemBuild™, and Xmath™ are trademarks of National Instruments Corporation.

Product and company names mentioned herein are trademarks or trade names of their respective companies.

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or `ni.com/patents`.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Conventions

The following conventions are used in this manual:

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
	This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.
bold	Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.
`monospace bold`	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.
`monospace italic`	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.
Platform	Text in this font denotes a specific platform and indicates that the text following it applies only to that platform.

Contents

Chapter 1
Introduction

Chapter 2
Template Programming Language

1 Introduction

The manual contains detailed instructions for using the template programming language (TPL) that can customize code generated by AutoCode or documentation generated by DocumentIt.

Manual Organization

This manual is organized as follows:

- Chapter 1, *Introduction*, provides an overview of the rapid prototyping concept, and how AutoCode and DocumentIt fit into it.
- Chapter 2, *Template Programming Language*, describes the template programming language and gives examples of its use.
- Chapter 3, *TPL Token Reference*, alphabetically lists and describes all TPL tokens.
- Appendix A, *AutoCode TPL Tokens Listed by SCOPE Class*, lists the AutoCode tokens by SCOPE class. This appendix provides locator tables that refer to Chapter 3, *TPL Token Reference*, for the details.
- Appendix B, *DocumentIt TPL Tokens Listed by SCOPE Class*, lists the DocumentIt tokens by SCOPE class. This appendix provides locator tables that refer to Chapter 3, *TPL Token Reference*, for the details.

This guide also has an *Index*.

Rapid Prototyping Concept

Conventional real-time system development usually takes place in stages, with separate tools for control design, software engineering, data acquisition, and testing. The MATRIXx product family integrates tools for each stage of system development into a single environment. This allows a design to move easily from one stage to the next, making it possible to create a working prototype early in the design process.

Within MATRIXx, a system model can be built, simulated, analyzed, tested, and debugged using SystemBuild and Xmath. Real-time code in a high-level language (C or Ada) for the model and design documentation

then can be generated automatically with AutoCode and DocumentIt, respectively. Figure 1-1 shows AutoCode and DocumentIt in the MATRIXx product line.

The generated application code can be evaluated on the host with SystemBuild simulation or run on NI real-time hardware for hardware in-the-loop testing. Finally, the generated application code can be cross-compiled and linked for implementation on an embedded processor. Figure 1-2 shows the code generation process. Documentation can be updated and automatically generated along each step of the process.

Figure 1-3 shows the document generation process.

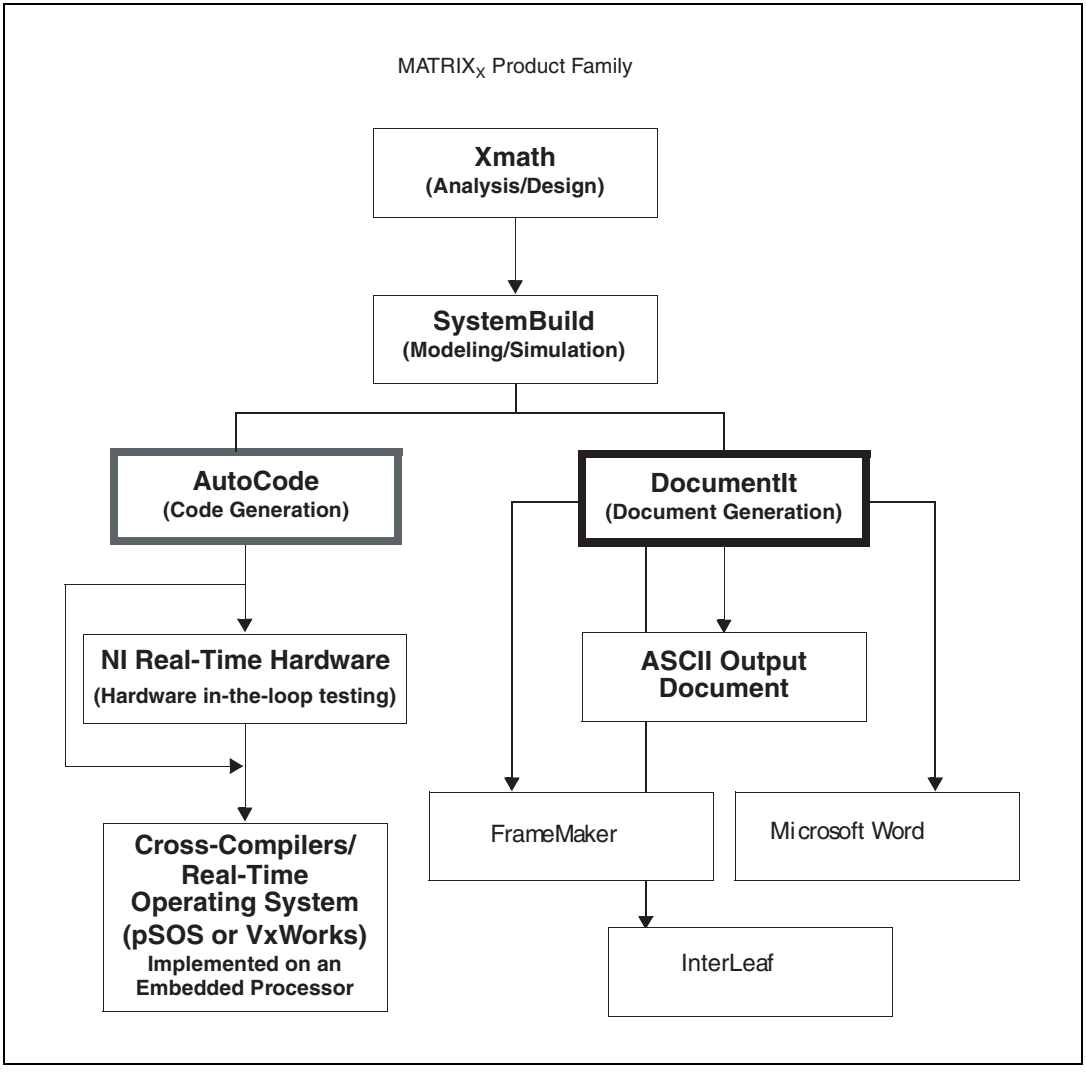

Figure 1-1. AutoCode and DocumentIt in the MATRIXx Product Line

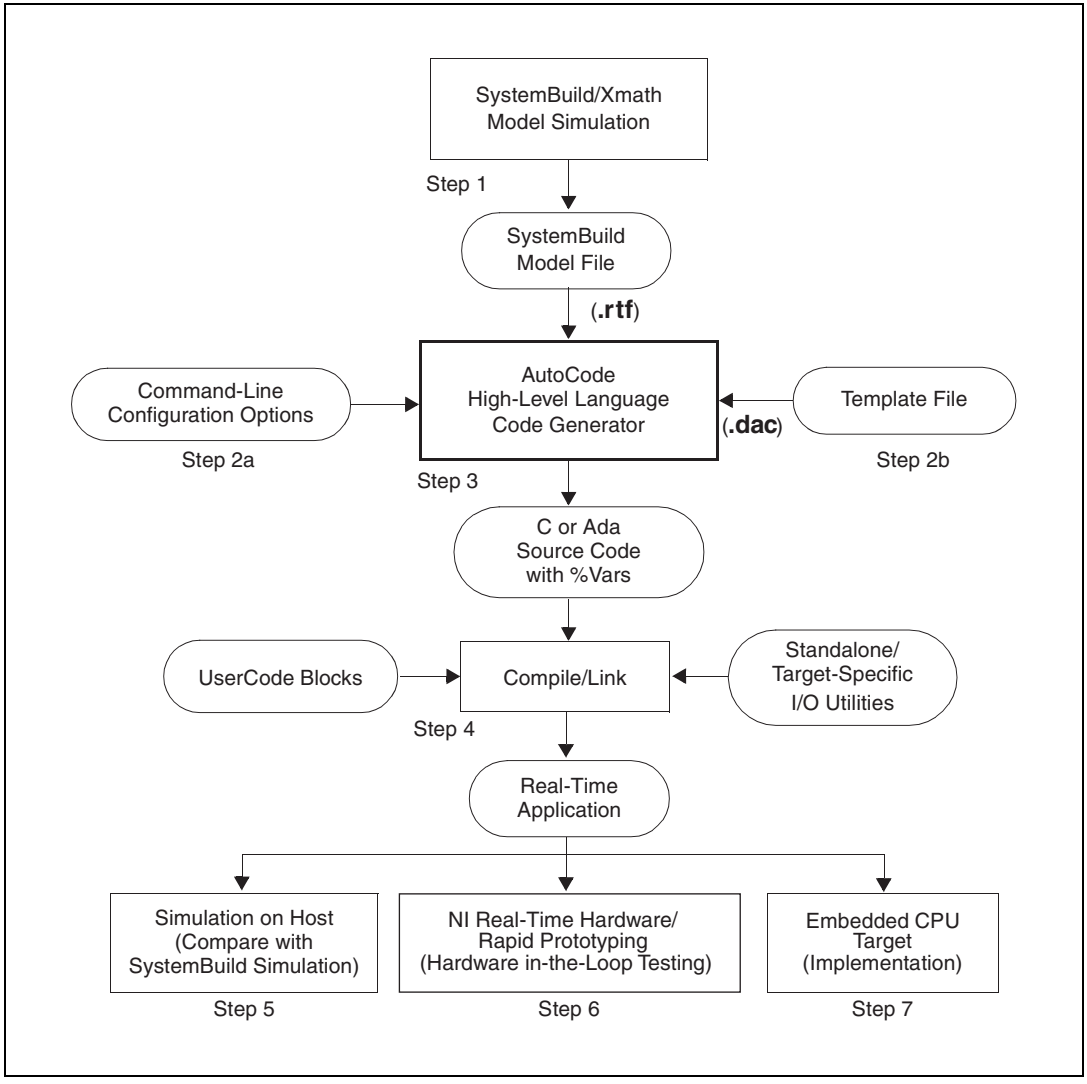

Figure 1-2. AutoCode Automatic Code Generation Process

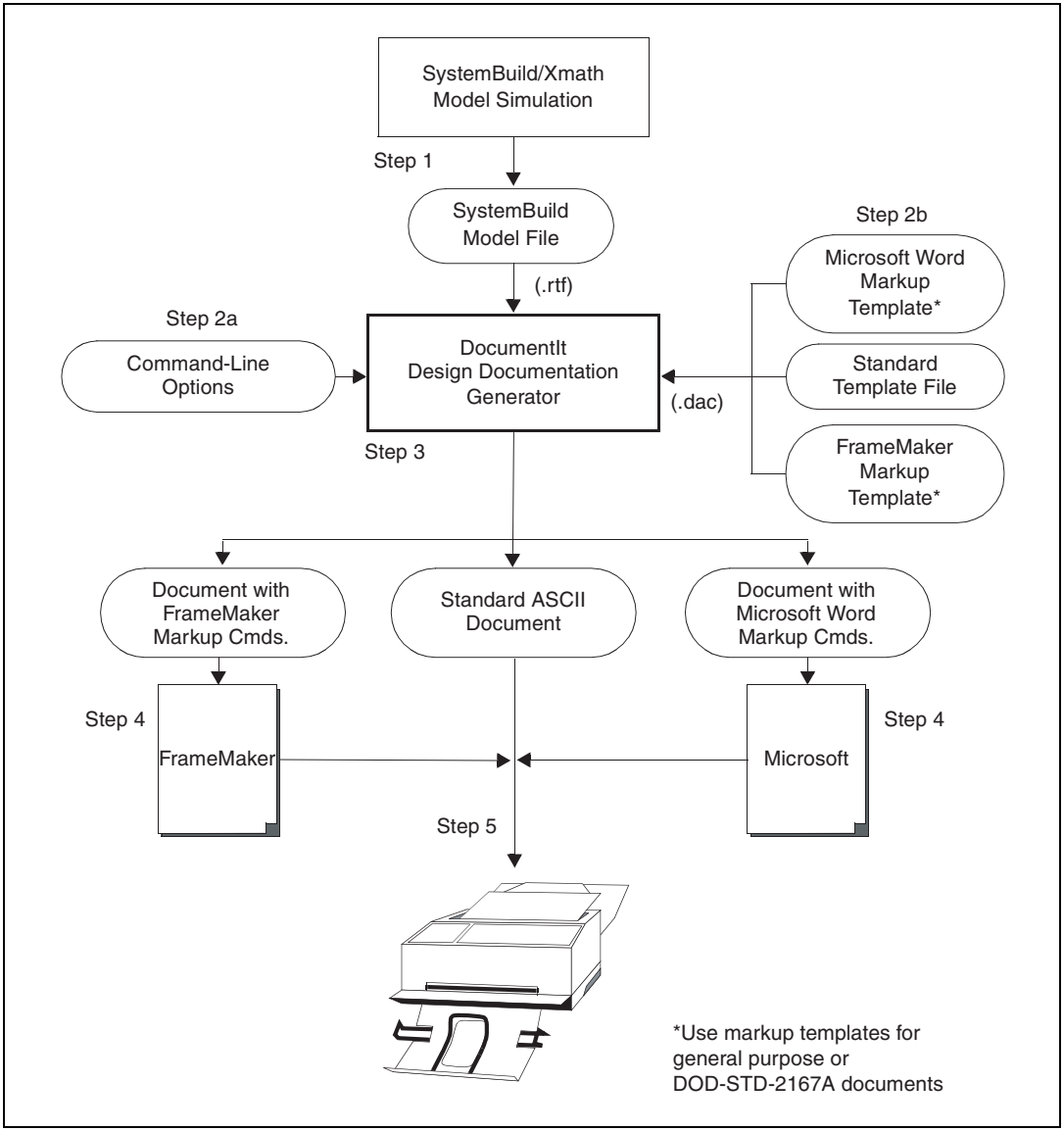

Figure 1-3. DocumentIt Automatic Documentation Generation Process

Customizing the Generated Code or Documentation

You can tailor your generated code or documentation by using the template programming language (TPL), provided in both AutoCode and DocumentIt.

You can customize the code for a wide variety of specialized purposes. Run-time parameterization can be programmed into the generated program. Also, configuration information can be entered when the real-time code is generated.

You can customize the documentation for a wide variety of specialized purposes. DocumentIt output in this case is an ASCII text file, which can be used by any commercially available text publishing software product. You can edit the file with the host editor or print it as a simple unformatted ASCII file.

Alternatively, if used with text publishing software, the template file not only contains DocumentIt parameters, but also can contain publishing software commands (for example, Microsoft Word RTF or FrameMaker MML), which DocumentIt writes directly to the output file. The DocumentIt output ASCII file then can be used as input to the text publishing software for automatic formatting.

Templates

This section describes the template options for both AutoCode and DocumentIt.

Templates—AutoCode

Templates serve as the front end to AutoCode. They determine completely what the output code should be for a given model (`.rtf` file) and command line options. The Template Programming Language (TPL) is used to specify the templates, which are merely TPL programs.

A TPL program (or a template source file) must be compiled into a binary file (direct access, or `dac` file) before it is executed by AutoCode to generate the output code. AutoCode checks for any syntactic or semantic errors while compiling a TPL program and produces a template `dac` file if no errors are found. This template `dac` file can then be used directly for further code generation sessions, thus saving time by avoiding recompilations.

Since the template dac files are machine (host) dependent, you may want to keep track of them.

Template Compile-and-Run Options—AutoCode

In the OS command line, AutoCode supports the following four options to compile and run your templates:

- `autostar -l hll rtf-file` looks for a `template-dacfile` in the current directory and if found executes it. If the language option is Ada (`-l a`), it looks for the dac-file named `ada_rt.dac`; for `C` it looks for `c_sim.dac`.
- `autostar -l hll -d template-dacfile rtf-file` executes the binary `template-dacfile`.
- `autostar -l hll -t template-srcfile rtf-file` compiles `template-srcfile` into a binary file (`dac` file) and executes it. The `dac` file is named with a `.dac` extension to `template-srcfile`.
- `autostar -l hll -t template-srcfile -d template-dacfile rtf-file` compiles `template-srcfile` into `template-dacfile` and executes `template-dacfile`.

Notice that the last two options support the compilation of the template source-file.

From the **Xmath Commands** window, AutoCode is invoked using the command `autocode`**.** The `-t` option is renamed as the `tplsrc` option and the `-d` option is renamed as the `tpldac` option. The default for the `tpldac` option is:

(for C)

`$ISIHOME/`*`platform`*`/case/ACC/templates/c_sim.dac`

(for Ada)

`$ISIHOME/`*`platform`*`/case/ACA/templates/ada_rt.dac`

Templates—DocumentIt

Templates determine completely what the output documentation will be for a given model (`.rtf` file). The Template Programming Language (TPL) is used to specify the templates. These are referred to as TPL programs.

A TPL program (or a template source file) must be compiled into a binary file (direct access, or `dac` file) before it is executed by DocumentIt to generate the output documentation. DocumentIt checks for any syntactic or semantic errors while compiling a TPL program and produces a template

dac-file if no errors are found. This template dac-file then can be used directly for further document generation sessions, thus saving time by avoiding recompilations.

Since the template dac files are machine-dependent, you may want to keep track of them.

Template Compile-and-Run Options—DocumentIt

From the operating system prompt, DocumentIt supports the following options to compile and run your templates:

- `autostar -doc rtf-file` looks for a `template-dacfile` named `documentit.dac` in the current directory and if found executes it.
- `autostar -doc -d template-dacfile rtf-file` executes the binary `template-dacfile`.
- `autostar -doc -t template-srcfile -d template-dacfile rtf-file` compiles `template-srcfile` into `template-dacfile` and executes `template-dacfile`.

Notice that the last two options support the compilation of the template source file.

From the **Xmath Commands** window, invoke DocumentIt using the `documentit` command. The `-t` option is renamed as the `tplsrc` option.

Related Publications

National Instruments provides a complete library of publications to support its products. In addition to this guide, publications that you may find particularly useful when using TPL include the following:

- *Xmath User Guide*
- *SystemBuild User Guide*
- *DocumentIt User Guide*
- *AutoCode User Guide*
- *AutoCode Reference*

2

Template Programming Language

This chapter describes the template programming language (TPL) and gives usage examples.

Introduction

TPL is designed for customizing and parameterizing AutoCode or DocumentIt output. Its support for model-related parameters, in addition to the normal programming language features, gives you an added flexibility in parameterizing the customized output. These parameters are evaluated by AutoCode or DocumentIt at run time for the specified `rtf` file in that session and are replaced by appropriate values in the TPL program.

Sample AutoCode C Template

The parameters are grouped in terms of concepts in the model such as block and SuperBlock, and TPL lets you refer to any specific instance of these model objects. This capability provides the means at code generation time to scope on the objects originally defined in the model. TPL also makes printing of text (source code, comments, or any sequence of ASCII characters) easy. Example 2-1 is an example C TPL program that generates a “Hello, World” C program.

Example 2-1 C example_c1.tpl Program

```
@/ TPL program for a "Hello, World" C program
@SEGMENT MAIN()@@
/* This C program prints "Hello, World". */
#include "stdio.h"
void main()
{
printf("Hello, World\n");
}
@ENDSEGMENT@
```

Running the AutoCode OS utility (autostar) for the .rtf file *any_model*.rtf compiles the template source file example_c1.tpl into example_c1.dac, executes the template dac file example_c1.dac, and produces the output in hello.c:

```
% autostar -l c -t example_c1.tpl -o hello.c any_model.rtf
```

The hello.c output file should contain the following lines:

```
/* This C program prints "Hello, World". */
#include "stdio.h"
void main()
{
printf("Hello, World\n");
}
```

Sample AutoCode Ada Template

Example 2-2 is an example Ada TPL program in a file named example_ada1.tpl. The TPL program generates a “Hello, World” Ada program in the specified output file specified in the command syntax.

Example 2-2 Ada example_ada1.tpl Program File

```
@/ TPL program for a "Hello, World" Ada program
@SEGMENT MAIN()@@
-- This Ada program prints "Hello, World". --
with TEXT_IO; use TEXT_IO;
procedure main is
begin
  put ("Hello, World");
  new_line;
end main;
@ENDSEGMENT@
```

Running autostar for the .rtf file *any_model*.rtf compiles the template source file example_ada1.tpl into example_ada1.dac, executes the template dac file example_ada1.dac, and produces the output in hello.a:

```
% autostar -l a -t example_ada1.tpl -o hello.a any_model.rtf
```

The output file hello.a should contain the following lines:

```
-- This Ada program prints "Hello, World". --
with TEXT_IO; use TEXT_IO;
procedure main is
begin
```

```
  put ("Hello, World");
  new_line;
end main;
```

Note The `rtf` file argument is mandatory even if this TPL program does not contain any model-related parameters. Running AutoCode with either of the example `dac` files with any `rtf` produces the same output shown above because the TPL programs are not using information about the model nor are they using any of the model code-generation capabilities of AutoCode.

Sample DocumentIt Template

Example 2-3 shows an example TPL program in a file named `example_doc1.tpl`. The TPL program generates a sample document in the output file specified in the command syntax (refer to Example 2-4).

Example 2-3 Example TPL Program (example_doc1.tpl)

```
@SEGMENT MAIN()@
************************************************************************
|              DocumentIt (TM) Document Generator V7.x                 |
|                                                                      |
|          National Instruments Corporation, AUSTIN, TEXAS             |
|                                                                      |
************************************************************************
Filename                : @currentfname_s@
Generated on            : @currentdate_s@
Dac file created on     : @dac_createdate_s@

This is a Software Design document generated using DocumentIt

You can type messages like this, which will appear exactly in the output file
generated by DocumentIt, or you can use the parameters given. A parameter can
be a built_in one or user defined using user_param.

The TPL files are necessary when you generate any documentation. TPL files
are available for general purpose and 2167A standard documents using
FrameMaker and Microsoft Word RTF.
@ENDSEGMENT@
```

Running `autostar` for an `rtf mymodel.rtf` compiles the template source file `example_doc1.tpl` into `example_doc1.dac`, executes the template `dac` file `example_doc1.dac`, and produces the output in *`mymodel`*`.doc`:

```
% autostar -doc -t example_doc1.tpl mymodel.rtf
```

The output file *mymodel*.doc should contain the lines shown in the example below.

Example 2-4 DocumentIt Output from Example TPL Program (example_doc1.doc)

```
**************************************************************************
|              DocumentIt (TM) Document Generator V7.x                    |
|                                                                         |
|          National Instruments Corporation, AUSTIN, TEXAS                |
|                                                                         |
**************************************************************************

Filename                : mymodel.doc
Generated on            : Fri Dec  3 12:46:14 1998
Dac file created on     : Fri Dec  3 12:46:13 1998

This is a Software Design document generated using DocumentIt

You can type messages like this, which will appear exactly in the output file
generated by DocumentIt, or you can use the parameters given. A parameter can
be a built_in one or user defined using user_param.

The TPL files are necessary when you generate any documentation. TPL files
are available for general purpose and 2167A standard documents using
FrameMaker and Microsoft Word RTF.
```

Note The `rtf` file argument is mandatory even if this TPL program does not contain any model-related parameters. Running DocumentIt with the example `dac` file `example_doc1.dac` produces the same output as shown if the same `rtf` file is used.

Syntax and Semantics of TPL

This section describes how to write a TPL program.

Basic Structure

This section describes the overall structure of a TPL program.

Main Segment

A TPL program consists of segments which are equivalent to functions in C or Ada. A segment (or function) contains statements that specify the computing operations to be done, as well as parameters and variables that store values used during the computation. You can give a segment any name (except the reserved words; refer to the *TPL Language Keywords* section), but a segment named `MAIN` must be present in any TPL program, because the TPL program starts its execution at the beginning of `MAIN`.

Since TPL is case sensitive, the name MAIN and all other reserved words must be specified with their exact case. The MAIN segment calls other segments which can be library segments or user-written TPL segments. A segment can be defined to take arguments by specifying them within parentheses. As in the previous examples, MAIN is always defined as a segment that expects no arguments, which is indicated by the empty parentheses. The statements of a segment are enclosed between the SEGMENT statement and ENDSEGMENT statement.

Both the SEGMENT statement and the ENDSEGMENT statement are delimited by the symbol @ which is a special symbol used in general to delimit TPL statements. It also is used as an escape character for @ and newline characters.

Comments

The symbol @/ specifies a single line comment, which forces the TPL parser to skip the rest of the line including the newline character at the end. Multi-line comments are specified by the delimiters @* and *@. Comments are ignored and are not printed in the output.

Plain Text

Plain text or newline characters by themselves, which are not delimited by @, serve as print statements, because they are printed in the output file verbatim (refer to Example 2-4). An extra @ at the end of the MAIN segment declaration statement escapes the newline character. Consequently, that newline is not printed in the output. This is the case with all the TPL statements delimited by @, which are followed by newline characters. These newline characters are printed in the output if they are not explicitly escaped by @. All the spaces, tabs, and newlines that are outside the segment body or that are within the TPL statement delimiters are skipped and are not printed in the output.

Whitespaces and Alignment

Within TPL, whitespace is significant in that outside of a `@` delimited statement, the whitespace will appear in the generated file. However, you can use white spaces at the beginning of a `@` delimited expression to align statements for easier reading. Also, the `@* *@` comment pair can be used to align plain text within the program without causing extra spaces in the generated output. For example:

```
@LOOPP i=0, i lt 5, i = i plus 1@@
@  LOOPP j=0, j lt 5, j = j plus 1@@
@*  *@Value of (i): @i@
@*  *@Value of (j): @j@
@  ENDLOOPP@@
@ENDLOOPP@@
```

Types, Operators, Expressions

TPL supports variables, parameters, and constants. These can be used in expressions, along with the operators specifying which operations must be done on these basic objects, to produce new values or used in print statements to generate ASCII output of their values.

Scalar and Array Variables

Variables can be either scalar or an array of either an integer, string or double data type. Variables names must start with a letter or an underscore; they can be made up of letters, digits, and underscores. There is no limit on length of the name, but you cannot use TPL reserved names as variable names. Variables can be freely written into during program execution and they preserve their values as long as they are in scope. Variables must be declared before they are used. They can be declared either in global scope so that they are visible across the TPL segments or in local scope within a segment.

The syntax for the variable declarations is:

```
@INT variable-list@
@STRING variable-list@
@DOUBLE variable-list@
```

where *variable-list* is the list of variable names separated by commas. Array variables are declared by appending the size within brackets `[]` to the variable name, similar to a C array declaration. Arrays are 0-based and statically sized.

Global variables are declared by placing their declaration statements outside of a segment. They become visible from the point at which they are declared and are available until the end of the program. Local variable declarations must immediately follow the parentheses and precede the ending @ delimiter of the SEGMENT declaration statement. The @ delimiters are not required for the local declarations since they are enclosed within the SEGMENT declaration statement, which has to be delimited by @s. Local variables are visible only within the segment in which they are declared. They lose their meaning after the segment is exited.

A TPL program can have more than one global declaration statement and all global variables are guaranteed to be initialized to zero. A program, however, can have only one local variable declaration. NI recommends a blank line to separate different data type declarations as shown in Example 2-5.

Example 2-5 Variable Declarations

```
@INT alpha, beta, gamma[10]@    @/ global declaration
@STRING up, down_under@         @/ global declaration

@SEGMENT MAIN() INT i,j DOUBLE delta @  @/local
declarations
@gamma[0] = alpha plus beta times delta@
@up = "Upwardly mobile"@
@ENDSEGMENT@
```

Template Tokens (Parameters)

Template tokens, also known as parameters, are initialized once during run time as specified by the model and configuration switches for the current session. Unlike variables, you cannot write into parameters. Refer to Chapter 3, *TPL Token Reference*, for a complete listing of template tokens. Each parameter has a suffix that follows the last underscore in the parameter name; the suffix indicates parameter data type.

Parameters include the following data types:

- integer (_i)
- real (double-precision) (_r)
- Boolean (_b)
- string (_s)
- list of integers (_li)

- list of reals (double-precision) (`_lr`)
- list of Booleans (`_lb`)
- list of strings (`_ls`)

Note TPL supports mixed integer and real (double) arithmetic.

List parameters must be dereferenced before they can be used in expressions. All the list parameters can be dereferenced by the operator `[]`, which takes an integer expression as the index. All the list parameters start with the index 0. List parameters are usually associated with corresponding integer type size parameters denoting the sizes of the lists. Dereferencing the list parameters out of bounds would result in a fatal error while generating code. List parameters, when printed without de referencing, result in a string of list elements separated by commas in the output. This allows easy initialization of arrays in the generated code.

Scope Classes

In addition to the data types, the parameters also are grouped by abstract classes in the model. Each parameter belongs to a specific scope class. A scope class represents an abstract entity in the model. There are seven distinct scope classes, each class supporting a unique list of parameters.

The scope classes are:

- `SYSTEM`—Represents the whole model
- `DATASTORE`—Represents all DataStores in the model
- `PROCESSOR`—Represents all processors in the system (AutoCode only)
- `SUBSYSTEM`—Represents all tasks in the system (AutoCode only)
- `PROCEDURE`—Represents all procedures in the system (AutoCode only)
- `SUPERBLOCK`—Represents all SuperBlocks in the model (DocumentIt only)
- `BLOCK`—Represents all blocks in the currently scoped (parent) SuperBlock (DocumentIt only)

All of these classes can have multiple instances (objects) in the model except the `SYSTEM` class.

Before a parameter can be used in the TPL program, you must explicitly instantiate the scope class with a specific instance ID. This process is called

instantiation or *scoping* of the scope class. This instantiation needs to be done only once for all the parameters in this scope class for a particular instance of the class. The values of the parameters belonging to this scope class are preserved throughout the program, even across the segments, until another instantiation is done for another instance of the same scope class.

To specify this instantiation, TPL provides a `SCOPE` statement. The scope statement instantiates a scope class for a particular instance, thus loading all the parameters belonging to this scope class at runtime with values computed from the specified instance or object. The syntax of the scope statement is:

```
@SCOPE scope-class-name expression@
```

where `scope-class-name` represents one of the previously listed scope class names and `expression` should evaluate to zero or a positive integer (ID) representing the instance or object of the scope-class.

When creating a custom template, include a `@SCOPE PROCESSOR 0@` statement to initialize all system tokens.

Note If you do not include a `@SCOPE PROCESSOR 0@` statement, AutoCode will fail with an assertion error when you call any TPL library function.

TPL uses the following numbering scheme for the unique identification of models.

Table 2-1. ID Ranges by Object Class

Class	ID Range
`SYSTEM`	ID is always 0. All the parameters belonging to the `SYSTEM` class are automatically loaded and explicit instantiation by the user is not required to use any of the parameters of the `SYSTEM` class. For example, `ntasks_i`, which is a parameter of integer data type belonging to the `SYSTEM` scope class, is always loaded with the value equal to the total number of tasks in the model.
`DATASTORE`	0 to the number of datastores in the model minus 1.
`PROCESSOR`	0 to the number of processors in the system minus 1.
`SUBSYSTEM`	1 to the number of tasks in the system (the ID 0 is reserved for the scheduler ID).
`PROCEDURE`	0 to the number of procedures in the model minus 1.

Table 2-1. ID Ranges by Object Class (Continued)

Class	ID Range
SUPERBLOCK	0 to the number of SuperBlocks in the model minus 1.
BLOCK	0 to the number of blocks in the parent SuperBlock minus 1. The SUPERBLOCK class must be scoped before a BLOCK can be scoped, because the block instance numbering is relative to the parent SUPERBLOCK.

For example, a model might have two tasks and five procedures, implying that the SUBSYSTEM class (AutoCode) can have two instances with IDs 1, 2 and the PROCEDURE class can have five instances with IDs 0, 1, 2, 3, 4. To specify the parameter numxs_i, which represents the number of states in any task (subsystem)/procedure, to have a value equal to the number of states in subsystem 1 requires the following scope statement before the statement which uses numxs_i:

```
@SCOPE SUBSYSTEM 1@
```

If you want to refer to a model object of a specific name, but you do not know the ID of the object, loop on the number of objects, scope for each object using the loop index (thus loading its name parameter), compare the name parameter against a specific name using the STRCMP( ) function (a TPL library function), and test its return value (which is 1 if they are equal, 0 otherwise). If the return value is 1, then the object in the current scope has the name you wanted. Notice that if the names are not unique there may be other objects with the same name (for example, two blocks in a SuperBlock can have the same name). Then, by continuing the previous loop, you can access the other objects that have the same name progressively.

The complete list of the parameters belonging to different scope classes and their purposes are provided in Chapter 3, *TPL Token Reference*, through Appendix B, *DocumentIt TPL Tokens Listed by SCOPE Class*. Chapter 3, *TPL Token Reference*, lists and describes the TPL parameters. Appendix A, *AutoCode TPL Tokens Listed by SCOPE Class*, lists the AutoCode parameters by SCOPE class, cross-referencing the parameters to Chapter 3, *TPL Token Reference*, where they are completely described. Appendix B, *DocumentIt TPL Tokens Listed by SCOPE Class*, lists the DocumentIt parameters by SCOPE class, cross-referenceing the parameters to Chapter 3, *TPL Token Reference*, where they are completely described. Notice that the SUBSYSTEM and PROCEDURE classes contain the same list of parameters. Consequently, only one scope is valid at a time; that is, you cannot refer to a SUBSYSTEM object and a PROCEDURE object at the same

time. As a performance note, since the SCOPE statement loads all the parameters in the class for a given instance, you may want to localize your code to use/reuse all of these parameters for this instance and minimize the number of times you have to use SCOPE.

Constants

Constants can be of any one of integer, double, string, or text types. Integer and double constants are as expected, like 5 or 3.14. A string constant is a sequence of characters delimited by double quotes (such as "output.c"). Double quote and newline characters, if present within a string, must be escaped with a backslash as \" and \n, respectively.

String constants are used as arguments to special library functions, which take strings as arguments such as FILEOPEN and STRCMP( ). String constants are limited to 255 characters in length.

The text constants are either the verbatim text (any sequence of ASCII characters except newline) or newline characters within the segment body, which are not surrounded by @. They are printed as is, in the output file.

The @ character itself can be printed by using an extra @ as the escape character (for example, @@). If you do not want to print the newline character, you must escape the newline character by preceding it with an @. This feature also is useful in breaking up long text into multiple lines in the template source file, but you want the text to be printed without any line breaks in the output file.

Operators

TPL operators can be applied to variables, parameters, and constants to create TPL expressions. Operators can be applied to expressions as well. They can be classified as arithmetic, relational, logical, and assignment operators. Additionally, there is also a string concatenation left-associative operator, cat, which operates on string constants or parameters of string data type. The cat operator can appear in any string expression (for example, a FILEOPEN or STRCMP statement).

Arithmetic operators are binary times, div, plus, minus, unary minus, mod, shr, and shl. Their behaviors are self-explanatory with the standard meanings (refer to Table 2-2).

Table 2-2. Arithmetic Operators

Arithmetic Operator	Meaning
`plus`	addition
`minus`	subtraction and unary negation
`times`	multiplication
`div`	division
`mod`	modulus division
`shl`	logical shift left
`shr`	logical shift right

The binary `plus` and `minus` have the same precedence, which is lower than the precedence of `times`, `div`, and `mod`, which is in turn lower than unary `minus`. The logical shift operators have the lowest precedence of the arithmetic operators. For example:

```
5 plus 2 times 3     evaluates to 11 and not 21
```

Relational operators are shown in Table 2-3. Each relational operator tests if the expression on its left meets the criterion indicated in the table relative to the expression on its right. For example, `lt` tests if the expression on its left is less than the expression on its right. They have lower precedence than arithmetic operators. The `eq` and `neq` operators have lower precedence than `lt`, `gt`, `le`, and `ge`. For example:

```
i lt ntasks_i minus 1  is evaluated as:  i lt (ntasks_i
minus 1)
```

Logical operators are `and`, `or`, and unary negation `not`. Both `and` and `or` have lower precedence than relational operators. `and` has higher precedence than `or` and both expressions connected by `and` or `or` are evaluated whether or not the expression in the left evaluates to true. For example, you *cannot* use a single expression to guard against a division-by-zero by coding:

```
i neq 0 and 5 div i eq 2     results in division by zero
if i is 0
```

The unary negation operator `not` converts a zero operand into 1 and a nonzero operand into 0. For example:

```
not i    is equivalent to:   i eq 0.
```

Table 2-3. Relational Operators

Relational Operator	Meaning
`lt`	less than
`gt`	greater than
`le`	less than or equal to
`ge`	greater than or equal to
`eq`	equal to
`neq`	not equal to

Assignment operator = has the lowest precedence and is evaluated right to left. It is used to assign a value to a TPL variable on its left. It evaluates the expression to its right and assigns the value to the variable on the left. For example:

```
i=j=minus 9        assigns the value -9 to both i and j
```

Table 2-4 shows the summary of the precedence and associativity of TPL operators. Operators on the same row have the same precedence. The `( )` operator refers to the segment call and the `[]` operator refers to the list dereferencing operator.

Table 2-4. Operator Precedence from Highest to Lowest

Operators	Associativity
`( )`, `[ ]`, `cat`	left to right
`not`, `minus` (unary)	right to left
`times`, `div`, `mod`	left to right
`plus`, `minus`	left to right
`shr`, `shl`	left to right
`lt`, `gt`, `le`, `ge`	left to right
`eq`, `neq`	left to right
`and`	left to right
`or`	left to right
`=`	right to left

Expressions

Expressions are formed by combining variables, constants, and parameters using the TPL operators. Expressions can be included in parentheses to give them highest precedence of evaluation. A `SEGMENT` call by itself forms an expression the value of which is the integer result returned by the segment. If the segment does not `RETURN` any value, the result is undefined. An assignment expression such as `i=i plus 1` first evaluates `i plus 1`, stores the value in `i`, and then stores the value of `i` as the result in a temporary variable. Mixed arithmetic between integers and doubles is supported with type conversions implicitly done based on the data types of the variables.

Complete C and Ada Examples

Example 2-6 (for C) and Example 2-7 (for Ada) illustrate some of the features just discussed. The program uses `SCOPE` and `LOOPP` statements in conjunction with the user-defined `gen_task_defn` segment to print multiple C functions (or Ada procedures) and the calls to those C functions (or Ada procedures).

Example 2-6 C example_c2.tpl Program

```
@* This template outputs a "Hello, World" C program which prints multiple
times "Hello, World". It generates as many C functions as the number of tasks
in the model, with each function printing "Hello, World" as many times as the
number of outputs of the respective task object in the model.*@

@SEGMENT MAIN() INT i@@
/* This C program prints multiple "Hello, World".*/
#include "stdio.h"

@/ Generate all task functions
@LOOPP i=1, i le ntasks_i, i=i plus 1@@
@  gen_task_defn(i)@@/ Call the segment to generate task i
@ENDLOOPP@@

@*Generate main function with calls to the task functions.*@
void main()
{
@LOOPP i=1, i le ntasks_i, i=i plus 1@@
    subsys_@i@();
@ENDLOOPP@@
}
```

```
@ENDSEGMENT@@/  End of MAIN


@/ Definition of the segment gen_task_defn()
@SEGMENT gen_task_defn(id)@@
@/Scope the SUBSYSTEM class for id
@SCOPE SUBSYSTEM id@

void subsys_@id@() {
int i;      @*Note that this i is merely part of the text*@
for(i=0; i<@outputs_i@; i++)
printf("Task @id@: Hello, World\n");
}
@ENDSEGMENT@
```

Example 2-7 Ada example_ada2.tpl Program

```
@* This template outputs a "Hello, World" Ada program which prints multiple
times "Hello, World". It generates a package with as many Ada procedures as
the number of tasks in the model, with each procedure printing "Hello, World"
as many times as the number of outputs of the respective task object in the
model.*@

@SEGMENT MAIN() INT i@@
-- Prints "Hello, World" multiple times. --
@/ Generate all procedures in a package
package subsystems is
@LOOPP i=1, i le ntasks_i, i=i plus 1@@
    procedure subsys_@i@;
@ENDLOOPP@@
end subsystems;

with TEXT_IO; use TEXT_IO;
package body subsystems is@
@INDENT 4@
@LOOPP i=1, i le ntasks_i, i=i plus 1@@
@   gen_task_defn(i)@@/Call the segment to generate task i
@ENDLOOPP@@
@INDENT 0@
end subsystems;

@*Generate main procedure with calls to the procedures *@
with subsystems;
procedure main is
```

```
begin
@LOOPP i=1, i le ntasks_i, i=i plus 1@@
  subsystems.subsys_@i@;
@ENDLOOPP@@
end main;
@ENDSEGMENT@@/    End of MAIN

@/ Definition of the segment gen_task_defn()
@SEGMENT gen_task_defn(id)@@
@SCOPE SUBSYSTEM id@
procedure subsys_@id@ is
@/ This loop index i is merely part of the text.
begin
  for i in 1..@outputs_i@ loop
     put_line ("Task @id@: Hello, World");
  end loop;
end subsys_@id@;
@ENDSEGMENT@
```

This TPL program, `example2.tpl`, when compiled and run for a model that has two tasks with three and seven outputs each, generates the output file shown in Example 2-8 (for C) and Example 2-9 (for Ada).

Example 2-8 C Output File

```
/* This C program prints multiple "Hello, World".*/
#include "stdio.h"

void subsys_1() {
int i;
for(i=0; i<3; i++)
printf("Task 1: Hello, World\n");
}

void subsys_2() {
int i;
for(i=0; i<7; i++)
printf("Task 2: Hello, World\n");
}

void main()
{
    subsys_1();
    subsys_2();
}
```

Example 2-9 Ada Output File

```
-- This Ada program prints "Hello, World" --
-- multiple times. --
package subsystems is
  procedure subsys_1;
  procedure subsys_2;
end subsystems;

with TEXT_IO; use TEXT_IO;
package body subsystems is
   procedure subsys_1 is
   begin
     for i in 1..3 loop
       put_line ("Task 1: Hello, World");
     end loop;
   end subsys_1;

   procedure subsys_2 is
   begin
     for i in 1..7 loop
       put_line ("Task 2: Hello, World");
     end loop;
   end subsys_2;
end subsystems;

with subsystems;
procedure main is
begin
  subsystems.subsys_1;
  subsystems.subsys_2;
end main;
```

The LOOPP statement and the SEGMENT definition are explained in detail in the sections that follow.

Complete DocumentIt Program

Example 2-10 illustrates some of these features for DocumentIt. The program uses a global variable called id, a local variable called i (loop counter in MAIN) and the parameters nsupblks_i (SYSTEM parameters). It uses the SCOPE and LOOPP statements in conjunction with the user-defined gen_doc_info segment to print SuperBlock information. The program is self-commenting.

Example 2-10 DocumentIt Example TPL Program to Print SuperBlock Information

```
@* This template generates a document which will describe each of the input(s)
and output(s) of all SuperBlocks within the model. In addition, the document
will contain a listing of each SuperBlock and all referenced SuperBlocks
within the SuperBlock. *@

@INT id@ @/ Global variable used to denote currently scoped SuperBlock

@SEGMENT MAIN() INT i@
@/ Generate all SuperBlock details
@LOOPP i = 1, i lt nsupblks_i, i = i plus 1@@
@  gen_doc_info()@@/ Call the segment to generate SuperBlock i
@ENDLOOPP@

@all_refs()@

@ENDSEGMENT@

@/ Definition of the segment gen_doc_info()
@SEGMENT gen_doc_info() INT j@@
@SCOPE SUPERBLOCK id@@/ Scope to the SuperBlock

------------------------------------------------------------
@sb_name_s@ Inputs (total #@num_sb_in_i@)
------------------------------------------------------------
@LOOPP j = 0, j lt num_sb_in_i, j = j plus 1@@

SB_EXTIN_ID_LS = @sb_extin_id_ls[j]@
SB_EXTIN_DSC_LS = @sb_extin_dsc_ls[j]@
SB_EXTIN_TYP_LS = @sb_extin_typ_ls[j]@
SB_EXTIN_UNIT_LS = @sb_extin_unit_ls[j]@
SB_EXTIN_MINV_LR = @sb_extin_minv_lr[j]@
SB_EXTIN_MAXV_LR = @sb_extin_maxv_lr[j]@
SB_EXTIN_ACCU_LR = @sb_extin_accu_lr[j]@

@ENDLOOPP@@
------------------------------------------------------------
@sb_name_s@ Outputs (total #@num_sb_out_i@)
------------------------------------------------------------
@LOOPP j = 0, j lt num_sb_out_i, j = j plus 1@

SB_EXTOUT_ID_LS = @sb_extout_id_ls[j]@
SB_EXTOUT_DSC_LS = @sb_extout_dsc_ls[j]@
SB_EXTOUT_TYP_LS = @sb_extout_typ_ls[j]@
SB_EXTOUT_UNIT_LS = @sb_extout_unit_ls[j]@
SB_EXTOUT_MINV_LR = @sb_extout_minv_lr[j]@
SB_EXTOUT_MAXV_LR = @sb_extout_maxv_lr[j]@
```

```
SB_EXTOUT_ACCU_LR = @sb_extout_accu_lr[j]@
@ENDLOOPP@@

@ENDSEGMENT@

@SEGMENT all_refs() INT i,j@
================================================
SuperBlock References
================================================
@LOOPP i = 0, i lt nsupblks_i, i = i plus 1@@
@   SCOPE SUPERBLOCK i@@
@   INDENT 0@
@**@SuperBlock: @sb_name_s@@
@   INDENT 10@
@   LOOPP j = 0, j lt num_blks_in_sb_i, j = j plus 1@@
@      SCOPE BLOCK j@@
@      IFF is_sb_b@@/ is the block a superblock reference?
@*       *@@blk_name_s@
@      ENDIFF@@
@   ENDLOOPP@@
@ENDLOOPP@@
================================================
@ENDSEGMENT@
```

When the TPL program in the Example 2-10 is compiled and run for a model, it generates a document with input/output information of all SuperBlocks as shown in Example 2-11.

Example 2-11 Output Generated by example2.tpl

```
-----------------------------------------------------
mux3 Inputs (total #2)
-----------------------------------------------------

SB_EXTIN_ID_LS = reduce speed
SB_EXTIN_DSC_LS = logical signal to reduce speed
SB_EXTIN_TYP_LS = LOGICAL
SB_EXTIN_UNIT_LS =
SB_EXTIN_MINV_LR = 0.00000
SB_EXTIN_MAXV_LR = 0.00000
SB_EXTIN_ACCU_LR = 0.00000

SB_EXTIN_ID_LS = increase speed
SB_EXTIN_DSC_LS = logical signal to increase speed
SB_EXTIN_TYP_LS = LOGICAL
SB_EXTIN_UNIT_LS =
SB_EXTIN_MINV_LR = 0.00000
```

```
SB_EXTIN_MAXV_LR = 0.00000
SB_EXTIN_ACCU_LR = 0.00000

------------------------------------------------------------
mux3 Outputs (total #1)
------------------------------------------------------------

SB_EXTOUT_ID_LS = target acceleration
SB_EXTOUT_DSC_LS = desired vehicle acceleration
SB_EXTOUT_TYP_LS = DOUBLE
SB_EXTOUT_UNIT_LS = none
SB_EXTOUT_MINV_LR = -1.00000
SB_EXTOUT_MAXV_LR = 1.00000
SB_EXTOUT_ACCU_LR = 2.00000

------------------------------------------------------------
Cruise Control Logic SB Inputs (total #1)
------------------------------------------------------------

SB_EXTIN_ID_LS = speed error
SB_EXTIN_DSC_LS =
SB_EXTIN_TYP_LS = DOUBLE
SB_EXTIN_UNIT_LS =
SB_EXTIN_MINV_LR = 0.00000
SB_EXTIN_MAXV_LR = 0.00000
SB_EXTIN_ACCU_LR = 0.00000

------------------------------------------------------------
Cruise Control Logic SB Outputs (total #2)
------------------------------------------------------------

SB_EXTOUT_ID_LS = reduce speed
SB_EXTOUT_DSC_LS = logic signal to reduce vehicle speed
SB_EXTOUT_TYP_LS = LOGICAL
SB_EXTOUT_UNIT_LS =
SB_EXTOUT_MINV_LR = 0.00000
SB_EXTOUT_MAXV_LR = 0.00000
SB_EXTOUT_ACCU_LR = 0.00000
SB_EXTOUT_ID_LS = increase speed
SB_EXTOUT_DSC_LS = logic signal to increase vehicle speed
SB_EXTOUT_TYP_LS = LOGICAL
SB_EXTOUT_UNIT_LS =
SB_EXTOUT_MINV_LR = 0.00000
SB_EXTOUT_MAXV_LR = 0.00000
SB_EXTOUT_ACCU_LR = 0.00000

------------------------------------------------------------
Differentials Inputs (total #4)
------------------------------------------------------------
```

```
SB_EXTIN_ID_LS = desired speed
SB_EXTIN_DSC_LS = desired vehicle speed
SB_EXTIN_TYP_LS = DOUBLE
SB_EXTIN_UNIT_LS = KmH
SB_EXTIN_MINV_LR = 0.00000
SB_EXTIN_MAXV_LR = 160.000
SB_EXTIN_ACCU_LR = 0.0200000

SB_EXTIN_ID_LS = measured speed
SB_EXTIN_DSC_LS = measured vehicle speed
SB_EXTIN_TYP_LS = DOUBLE
SB_EXTIN_UNIT_LS = KmH
SB_EXTIN_MINV_LR = 0.00000
SB_EXTIN_MAXV_LR = 160.000
SB_EXTIN_ACCU_LR = 0.0200000

SB_EXTIN_ID_LS = measured acceleration
SB_EXTIN_DSC_LS = measured vehicle acceleration
SB_EXTIN_TYP_LS = DOUBLE
SB_EXTIN_UNIT_LS = KmH2
SB_EXTIN_MINV_LR = -120000.
SB_EXTIN_MAXV_LR = 120000.
SB_EXTIN_ACCU_LR = 0.0500000

SB_EXTIN_ID_LS = target acceleration
SB_EXTIN_DSC_LS = desired vehicle acceleration
SB_EXTIN_TYP_LS = DOUBLE
SB_EXTIN_UNIT_LS = N_A
SB_EXTIN_MINV_LR = -1.00000
SB_EXTIN_MAXV_LR = 1.00000
SB_EXTIN_ACCU_LR = 2.00000

------------------------------------------------------------
Differentials Outputs (total #2)
------------------------------------------------------------

SB_EXTOUT_ID_LS = speed error
SB_EXTOUT_DSC_LS = the difference between the measured
                                and desired speed
SB_EXTOUT_TYP_LS = DOUBLE
SB_EXTOUT_UNIT_LS =
SB_EXTOUT_MINV_LR = -160.000
SB_EXTOUT_MAXV_LR = 160.000
SB_EXTOUT_ACCU_LR = 2.00000

SB_EXTOUT_ID_LS = throttle actuator command
SB_EXTOUT_DSC_LS = limited throttle position command
SB_EXTOUT_TYP_LS = DOUBLE
```

```
SB_EXTOUT_UNIT_LS =
SB_EXTOUT_MINV_LR = 25.0000
SB_EXTOUT_MAXV_LR = 73.0000
SB_EXTOUT_ACCU_LR = 2.00000

================================================
SuperBlock References
================================================

SuperBlock: Controller Logic
         mux3
         Cruise Control Logic SB
         Differentials

SuperBlock: mux3

SuperBlock: Cruise ControlLogic SB

SuperBlock: Differentials

================================================
```

The LOOPP statement and the SEGMENT definition are explained in the following sections. The global variable ID can be avoided by passing it as an argument to the gen_doc_info function.

Statements

In general, the TPL statements are sequentially executed starting from the beginning of the MAIN segment. Control flow statements are used to specify the order in which the computations are performed. Following are all of the possible TPL statements and corresponding syntax. Notice that the *statement-list* wherever it appears, stands for one or more TPL statements. Multiple nesting of IFF, WHILE, and LOOPP statements is allowed.

- Assignment statement

  syntax:

  @*variable = expression*@

  Assigns the value of the result of the expression on the right of the assignment operator to the variable on the left.

- Scope statement

  syntax:

  @SCOPE *scope-class-name expression*@

  Used to instantiate the scope classes. Refer to the *Scope Classes* section for more detail.

- Assert statement

  syntax:

  `@ASSERT` *`expression`*`@`

  example:

  ```
  @ASSERT STRCMP(language_s, "C")@
  ```

  Asserts that the given expression is true; that is, it evaluates to a nonzero value. If it is true, the program execution then continues past this statement. If the expression evaluates to false or a zero value, the program is terminated and an assertion failure error message is displayed.

  The example asserts that the current AutoCode or DocumentIt session uses “C” as the language option. *`language_s`* is a `SYSTEM` class parameter, which is loaded with a string “C” if the selected language option is “C”. Placing this statement in a C language template prevents you from accidently using this C template when generating some other language such as Ada, by generating an assert failure error message. Similarly,

  ```
  @ASSERT STRCMP(language_s,"ADA")@
  ```

  in an Ada template asserts that the current AutoCode session uses Ada as the language option.

- `NULL` statement

  syntax:

  ```
  @NULL@
  ```

  No operation.

- Print statement

  syntax:

  ```
  plain-text
  newline
  @variable@
  @parameter@
  @parameter [expression]@
  @constant@
  ```

  Used to print plain text, newline, variable, or parameter. It also can be used to print tabs by specifying the column number (integer constant) delimited by `@`s.

- Indent statement

  syntax:

  ```
  @INDENT integer-constant@
  ```

  Sets the left margin for the output to the given number. The new margin setting takes effect after a new-line or carriage return is generated into the output file. When set, the margin stays constant until another indent statement is used to specify a different value.

- If statement

  syntax:

  ```
  @IFF expression@
  statement-list1
  @ENDIFF@
  or
  @IFF expression@
  statement-list1
  @ELSE@
  statement-list2
  @ENDIFF@
  or
  @IFF expression_1 @
  statement-list1
  @ELSEIFF expression_2 @
  statement-list2
  @ENDIFF@
  ```

  This is used for conditional execution of statements. The *expression* is evaluated; if it evaluates to a nonzero value, then *statement-list1* is executed. The execution then jumps to the next statement after ENDIFF. If it evaluates to a zero value, the execution jumps directly to the next statement after ENDIFF without executing *statement-list1*. The ELSE part is optional. If ELSE is present and the *expression* evaluates to a zero value, then *statement-list2* is executed rather than *statement-list1*. The execution then jumps to the next statement after ENDIFF. Nesting of IFFs is allowed and the ELSE always goes with the closest previous ELSE-less IFF.

- While statement

  syntax:

  ```
  @WHILE test-expression@
   statement-list
  @ENDWHILE@
  ```

The *test-expression* is evaluated first. If it is nonzero, the *statement-list* (body of the while) is executed; then, the *test-expression* is reevaluated. This cycle continues until the *test-expression* evaluates to zero, at which point the execution jumps to the next statement after ENDWHILE.

- Loop statement

  syntax:

  ```
  @LOOPP expression1, expression2, expression3@
  statement_list
  @ENDLOOPP@
  ```

  The *expression1* is evaluated first and only once when the statement is first encountered. *expression2* is then evaluated. If it is nonzero, the statement-list (body of the loop) is executed and then *expression3* is evaluated. Now, *expression2* is reevaluated. This cycle continues until *expression2* evaluates to zero, when the execution jumps to the next statement after ENDLOOPP. Usually, *expression1* does the initialization of a variable representing the loop counter, *expression2* tests for the terminal condition, and *expression3* increments the loop counter. Functionally equivalent to the C For statement.

- Continue statement

  syntax:

  ```
  @CONTINUE@
  ```

  Used to jump to the evaluation of *expression3* in a LOOPP or to the evaluation of the *test-expression* in a WHILE. If there are nested LOOPP or WHILE statements, it associates with the closest LOOPP or WHILE. Functionally equivalent to the C Continue statement.

- Break statement

  syntax:

  ```
  @BREAK@
  ```

  Used to break out of a LOOPP or a WHILE. If there are nested LOOPP or WHILE statements, it associates with the closest LOOPP or WHILE. Functionally equivalent to the C Break statement.

- Exit statement

  syntax:

  ```
  @EXIT@
  ```

  Used to exit the TPL program.

- Segment call statement

  syntax:

  `@segment-name (argument-list)@`

  where *segment-name* is the name of the user-defined or library segment and *argument-list* is the list of actual arguments separated by commas.

  Used to call the segments for their execution.

Note Only integers and doubles (reals) can be passed as actuals.

- Return statement

  syntax:

  `@RETURN expression@`

  Used to return a value from the segment. The program execution is transferred back to the caller.

Note Only an integer or double (real) can be returned by the segment. Use global variables for strings or arrays.

- File open and file close statements

  syntax:

  ```
  @FILEOPEN(filename, open-mode)@
  @FILECLOSE@
  ```

  Used to open and close a new file stream to which the output is redirected. These statements are explained in detail in the *File Include and Output* section.

- Segment and end segment statements

  syntax:

  ```
  @SEGMENT segment-name (params-list) decln-list@
  statement-list
  @ENDSEGMENT@
  ```

  Used to delimit the beginning and end of a segment.

  These statements are explained in detail in the *Segments (Functions) and Program Structure* section.

Segments (Functions) and Program Structure

syntax:

```
@SEGMENT segment-name (params-list) decln-list@
statement-list
@ENDSEGMENT@
```

where `params-list` denotes the list of formal arguments which are merely variable names separated by commas. The actuals are always passed by value to the segment.

Note The data type of a formal argument is assumed to be the data type of the actual. Only integer or double data type can be used.

The `decln-list` denotes zero or more declaration statements. These define local variables of the segment with a scope only within that segment. Some example segment specifications include:

```
@SEGMENT do_nothing()@
@SEGMENT pvar_data(varidx) INT i,j@
@SEGMENT subsys_control(ssid, action) INT i[10] DOUBLE
delta@
```

Segment (or Function) names must start with a letter or an underscore. They can be constituted of letters, digits, and underscores. All names are case sensitive. You can choose any name you want except the reserved names. Refer to the *TPL Language Keywords* section. A segment must be defined exactly once in the program if it is called anywhere. Forward declarations are not required. Consequently, the place of definition of the segment could follow the place where it is actually called. There are some supplied TPL library functions (listed in Table A-7, *AutoCode TPL Library Functions*, and in Table B-6, *DocumentIt Library Functions*, and explained in Chapter 3, *TPL Token Reference*) which, when called, emit code or document in the output file. There also are some library functions which do not emit code, but perform useful computations. There is no limit to the number of segments in a TPL program, but a segment named `MAIN` must be present in every TPL program.

All the functions are assumed to return a value. This value can be used directly in expressions or just ignored. The optional `RETURN` statement, when used, causes the program execution to exit the segment with the return value evaluated from the `RETURN` expression. If the `RETURN` statement is not explicitly specified, the control returns to the caller automatically, with an undefined return value when the `ENDSEGMENT`

is reached. If there is no caller (for example, the MAIN segment), the program terminates. TPL supports both direct and indirect recursion.

Any variable declared outside the SEGMENT - ENDSEGMENT delimiters are considered global and are visible throughout the rest of the program from the point at which they are declared. The local variables cannot hide global variables and they are visible only within the segment in which they are declared. All the global variables are automatically initialized to zero. The white spaces (spaces, tabs, and newlines) outside the segment body are ignored and are not printed in the output. It is an error to have uncommented text, other than white spaces outside the segment body.

User-Defined Parameters

You can declare your own parameters and use them in DocumentIt templates. For information on how to define user parameters (that is, parameters defined by you), refer to the *SystemBuild User Guide*. The syntax is similar to that of the TPL variable syntax.

The parameters can be of one of the following data types: string, Boolean, integer, and real. The suffixes on the user parameter name define the data type of the parameter: _s stands for sting, _b for Boolean, _i for integer, and _r for real.

To use the user parameter within a TPL source file, use the following syntax:

```
@temp = user_param("string1", "string2")@
```

where *temp* is a declared variable of the appropriate type, *string1* is the name of any user parameter declared in the currently scoped SuperBlock or block and *string2* can only be the BLOCK, SUPERBLOCK, or DATASTORE string denoting the scope classes.

An example for using this feature in a TPL file is as follows:

```
@STRING temp_str@@
@temp_str = user_param("OVERVIEW_s", "SUPERBLOCK")@@
```

The cruise.tpl file given in the $case/DIT/templates/ascii directory of the distribution illustrates the use of user_params.

To test for the existence of a user parameter, you can use the `user_param` built-in function within an If-statement. For example:

```
@IFF user_param("block_flowdata_r", "BLOCK")@@
     parameter exists!
@ELSE @@
     not here
@ENDIFF@@
```

Regardless of how you access the user parameter, if it does not exist, a warning message will be displayed.

Comment and User Parameter Editor

The Properties dialog box for a SuperBlock (or for a State Transition Diagram (STD) Bubble, or Transition) has a **Comments** tab. This tab runs a text editor that is used for creating and editing comments and user parameters. You also can launch other editors (for example, vi on UNIX or Notepad on a PC). For information on available editors and how to customize your list of editors, refer to the *SystemBuild User Guide*.

Comment Editor Limitations/Restrictions

- Do not use the | (vertical line) symbol in comment editor text as it is used for internal purposes.
- Do not use the ~ (tilde) symbol in comment editor text as it is used for internal purposes.
- Maker Markup Language (MML) statements begin with a left angle bracket (<) and end with a balancing right angle bracket (>).

  If your ASCII output files contain *less than* (<) or *greater than* (>) characters from your model, use your text editor to place a *slash* (\) character in front of these characters. This is required because these characters will otherwise be interpreted as invalid MML commands when the `*.mml` file is imported into FrameMaker.

  Special characters can be included in regular document text using a backslash (\). For example, `\t` represents a tab. `\n` represents a hard return, and `\xnn` represents a FrameMaker character code (a 1- or 2-digit hexadecimal number terminated by a space). You can use character codes in the ranges `\x20` to `\x7E` and `\x80` to `\xFE`. Other values are ignored.
- MML statement names are not case sensitive.

File Include and Output

TPL supports a preprocessor INC directive that would include a TPL file in another. All of the nested include files in the template source file are expanded out during the beginning of compilation. This gives you flexibility to manage large template files in smaller chunks and also to include other text/target code files such as handwritten C code automatically in every pass of the code generation.

syntax:

```
@INC file-name@
```

where file-name is a string constant (enclosed in double quotes). *file-name* is searched relative to the same directory as its parent file, unless the absolute path is specified.

TPL also supports FILEOPEN and FILECLOSE statements to open and close a new file stream to which the output is redirected. At any point in time, only one file stream is allowed open. When the program first starts its execution at MAIN, the output stream is set to the output file specified in the command syntax (main stream). If the named output file exists, its original contents are lost. After a FILEOPEN statement is executed, the output of the template processor is sent to the new file specified, until a FILECLOSE statement is executed. At this point, the output is appended to the mainstream, unless a new FILEOPEN statement is encountered again. In this case, output is directed to this file. If a FILEOPEN statement is reached before another already opened file is closed, the open file is automatically closed and the new file is opened.

syntax:

```
@FILEOPEN(filename, open-mode)
@FILECLOSE@
```

where filename is a string expression; that is, either a simple string constant or an expression formed with cat operator(s), string constants, string parameters and dereferenced stringlist parameters, and *open-mode* is either "new" or "append". The "new" mode destroys the old contents of the file and the "append" mode preserves it. There is a special filename string called "stdout", which can be used along with the "append" mode to write to the standard output. For example:

```
@FILEOPEN(modelname_s cat "_1.c", "new")@
```

This opens a new file named `cruise_1.c` if the `rtf filename` is `cruise.rtf`. The parameter `modelname_s` stores the prefix `cruise` in this case. The following command redirects the output to standard output until a `FILECLOSE` is reached:

```
@FILEOPEN("stdout", "append")@
```

Caution If `FILEOPEN` cannot open a file, TPL terminates the execution of the TPL file.

String Manipulation Functions

TPL provides several string manipulation functions:

- `STRCMP(`*`string1, string2`*`)`—Checks if strings are equal; returns 1 if equal, 0 if not.
- `STRLEN(`*`string1`*`)`—Returns the length of a string.
- `STRLOWER(`*`string1`*`)`—Returns the contents of a string with all alphabetical characters converted to lowercase.
- `STRNCMP(`*`string1, string2`*`)`—Checks if strings compare up to the length of the shorter string; returns 1 if matches, 0 if not.
- `STRSTR(`*`string1, string2`*`)`—Returns the first occurrence of one string in another.
- `STRUPPER(`*`string1`*`)`—Returns the contents of a string with all alphabetical characters converted to uppercase.

Examples of the string manipulation functions are as follows:

- `STRCMP(`*`string1, string2`*`)`

```
@SEGMENT test() STRING S1, S2@
@S1 = "Hello There"@
@S2 = "Hello There"@
@IFF STRCMP(S1,S2)@
Strings Match!
@ENDIFF@
@ENDSEGMENT@
```

  The strings are equal, so the message is printed.

- `STRLEN(`*`string1`*`)`

```
@SEGMENT test() STRING S1@
@INT len@
@S1 = "Hello There"@
String Length: len@
@ENDSEGMENT@
```

  A length of 11 is printed.

- STRLOWER(*string1*)

```
@SEGMENT test() STRING S1@
@S1 = "Hello There"@
@S1 = STRLOWER(S1)@
@S1@
@ENDSEGMENT@
```

  The string is converted to lowercase. The string **"Hello There"** is printed.

- STRNCMP(*string1*, *string2*)

```
@SEGMENT test() STRING S1, S2@
@S1 = "Hello There"@
@S2 = "Hello"
@IFF STRNCMP(S1, S2)@
'@S2@' found within '@S1@'
@ENDIFF@
@ENDSEGMENT@
```

  The strings are equal by the number of characters in S2, so the message is printed.

- STRSTR(*string1*, *string2*)

```
@SEGMENT test() STRING S1, S2@
@S1 = "Hello There"@
@IFF STRSTR(S1, "lo")@
@S2 = STRSTR(S1, "lo")@
@S2@
@ENDIFF@
@ENDSEGMENT@
```

  Assigns **"lo There"** to S2 and prints it. When used in an IFF statement, STRSTR returns 1 if string2 is within string1. When used in an assignment statement, the remaining substring is returned.

- STRUPPER(*string1*)

```
@SEGMENT test() STRING S1@
@S1 = "Hello There"@
@S1 = STRUPPER(S1)@
@S1@
@ENDSEGMENT@
```

  The string is converted to uppercase. The string **"Hello There"** is printed.

TPL Language Keywords

The following are TPL language keywords:

and	ASSERT	BLOCK
BREAK	cat	CONTINUE
DATASTORE	DOUBLE	div
ELSE	ELSEIFF	ENDIFF
ENDLOOPP	ENDSEGMENT	ENDWHILE
eq	EXIT	FILECLOSE
FILEOPEN	ge	gt
IFF	INDENT	INT
le	LOOPP	lt
minus	mod	neq
not	NULL	or
plus	PROCEDURE	PROCESSOR
RETURN	SCOPE	SEGMENT
shl	shr	STRING
SUBSYSTEM	SUPERBLOCK	SYSTEM
times	WHILE	

3

TPL Token Reference

This chapter describes the TPL tokens for AutoCode and DocumentIt. This includes:

- *Overview*
- *Special Type Encodings*
- *TPL Token Descriptions*

Overview

The tokens are listed in the following format.

name_of_the_token

Description: description of the token

Size: size of the token

Scope Class: scope class of the token

Category: AutoCode, DocumentIt, or both

The token suffixes listed in Table 3-1 indicate the type of data and whether the token is a scalar or a list of elements.

Table 3-1. Token Suffixes

Suffix	Data Type
`_s`	scalar string
`_b`	scalar Boolean
`_i`	scalar integer
`_r`	scalar real
`_ls`	list of strings
`_lb`	list of Booleans

Table 3-1. Token Suffixes (Continued)

Suffix	Data Type
_li	list of integers
_lr	list of reals

If you use AutoCode and DocumentIt, you can use DocumentIt TPL parameters in the AutoCode template by using the -doc option along with the -l option from the OS prompt. One benefit of this is that you can comment the generated code by specifying comments as properties of SuperBlocks and blocks in your model.

Keep the following in mind when you use these parameters:

- Parameters with sb apply to *global* SuperBlocks, not to SuperBlock block references.
- Parameters with ds apply to *global* DataStores, not to DataStore block references.
- Block parameters apply to plain blocks as well as State Transition Diagrams (STDs) or block references of SuperBlocks and DataStores.
- If a parameter applies to a SuperBlock, block, STD, or DataStore equally, this is pointed out in the text accompanying the block parameter description.

Special Type Encodings

Some tokens use a special encoding of information to represent a fixed-point type. The encoding reduces the total number of parameters to represent the type information. The syntax is as follows:

FXPTYPEID := *TypeName_Prefix_ID* * 128 + *radix*

FXPPACKEDID := *LEFT_FXPTYPEID* * 65536 + *RIGHT_FXPTYPEID*

The *TypeName_Prefix_ID* is the number corresponding to one of the fixed-point subtypes, such as RT_UBYTE. The *FXPPACKEDID* provides a compact representation of two types of *FXPTYPEID*, one called the LEFT type and the other the RIGHT type.

TPL Token Descriptions

actv_cond_ls

Description: Lists the activation conditions of the transitions for each of the bubbles in the currently scoped STD block. All values for the STD block can be retrieved by indexing this parameter within a loop on `num_bbls_in_std_i` while offsetting the index by using `bbl_trn_off_li`.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

ATOF(string)

Description: Convert the string into a double.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

ATOI(string)

Description: Convert the string into an integer.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

backgnd_procs_li

Description: Lists the global number for each background procedure SuperBlock. Use the global number for the argument to SCOPE PROCEDURE.

Size: `nbackgnd_procs_i`

Scope Class: System

Category: AutoCode

bbl_cmt_ext_ls

Description: Lists the extensions associated with the bubble comment fields.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

bbl_cmt_ls

Description: Lists the textual descriptions (comments) of all the bubbles of the currently scoped STD block. All values for the STD block can be retrieved by indexing this parameter within a loop on `num_bbls_in_std_i`.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

bbl_id_li

Description: Lists all bubble identifiers. Retrieve all identifiers by indexing this parameter within a loop on `num_bbls_in_std_i`.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

bbl_keyname_ls

Description: Lists the key names of all the bubbles of the currently scoped STD block. Retrieve all values for the STD block by indexing this parameter within a loop on `num_bbls_in_std_i`.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

bbl_name_ls

Description: Lists all bubble names in the currently scoped STD block. Retrieve all values for the STD block by indexing this parameter within a loop on `num_bbls_in_std_i`.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

bbl_trn_off_li

Description: The transitions for each of the bubbles in an STD block can be accessed by offsetting the total list of transition related values for each of the bubbles. Retrieve the values using `num_bbls_in_std_i`.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

bbl_typ_ls

Description: Lists the types of all the bubbles in the currently scoped STD block. All values for the STD block can be retrieved by indexing this parameter within a loop on `num_bbls_in_std_i`.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

blk_cmt_ext_s

Description: The extension associated with the block comment field.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_cmt_s

Description: The comments for the currently scoped block as entered in the Comment field of the SystemBuild block form. The content is unformatted and can contain any embedded DocumentIt `user_param` parameters.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_code_cmt_s

Description: A reserved variable that can be declared in the Comment field of the block/SuperBlock reference, State Transition Diagram, or DataStore reference. The comment associated with the parameter will be pulled out automatically during the code generation. In DocumentIt, this variable can be pulled out in block scope using `blk_code_cmt_s` instead of the `user-param` function. For information on how to use this token, refer to the *User-Defined Code Comments* section of the *AutoCode User Guide*.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_container_s

Description: The name of the container of the block.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_derived_id_i

Description: An internally generated counter for all SuperBlocks and DataStores instantiated in the current SuperBlock. Returns an error if the block is not a SuperBlock or DataStore. Use `is_sb_b` or `is_ds_b` to determine which blocks are SuperBlocks or DataStores, respectively.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_fld_data_ls

Description: Lists the block properties **Parameters** tab, followed by a colon, followed by the data for a field for all the fields of the currently scoped block.

Size: `num_blk_flds_i`

Scope Class: Block

Category: DocumentIt

blk_fld_keyname_ls

Description: Lists the block properties **Parameters** tab for all the fields of the currently scoped block.

Size: `num_blk_flds_i`

Scope Class: Block

Category: DocumentIt

blk_has_in_b

Description: Determines if the currently scoped block has inputs, since not all blocks have inputs.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_has_out_b

Description: Determines if the currently scoped block has outputs, since not all blocks have outputs.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_has_outdata_b

Description: Determines if a user filled out any of the fields of the **Outputs** tab for the currently scoped block.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_has_pars_b

Description: Determines if a block has parameters, since not all blocks have parameters.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_has_state_b

Description: Determines if the currently scoped block has states, since not all blocks have states.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_id_i

Description: The ID of the currently scoped block object.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_in_accu_lr

Description: Lists the accuracies or precisions for all the inputs of the currently scoped block. Obtained from the Output Accuracy field of the source block **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_accu_ls

Description: Lists the accuracies or precisions for all the inputs of the currently scoped block. Obtained from the Output Accuracy field of the source block **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_dsc_ls

Description: Lists the textual descriptions for all the inputs of the currently scoped block. Obtained from the Output Comment field of the source block **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_id_ls

Description: Lists the IDs for all the inputs of the currently scoped block. Obtained from the Output Label field of the source block **Outputs** tab. All input IDs for a block can be retrieved by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_maxv_lr

Description: Lists the maximum values for all the inputs of the currently scoped block. Obtained from the Output Maximum field of the source block **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_maxv_ls

Description: Lists the maximum values for all the inputs of the currently scoped block. Obtained from the Output Maximum field of the source block **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_minv_lr

Description: Lists the minimum values for all the inputs of the currently scoped block. Obtained from the Output Minimum field of the source block **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_minv_ls

Description: Lists the minimum values for all the inputs of the currently scoped block. Obtained from the Output Minimum field of the source block **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_mnem_ls

Description: Lists the mnemonic names for all the inputs of the currently scoped block. Obtained from the Mnemonic field of the source block **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_name_ls

Description: Lists the names for all the inputs of the currently scoped block. Obtained from the Output Name field of the source block **Outputs** tab. Retrieve all input names for a block by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_radix_li

Description: Lists the radix numbers for all the inputs of the currently scoped block. Obtained from the Radix field of the source block **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_srcchn_li

Description: Lists block channel number from which the currently scoped block gets its input data. This parameter can be indexed within a loop on `num_blk_in_i` to obtain all the source block channel numbers from which the currently scoped block gets its individual inputs. In other words, the source of the input of the currently scoped block is the `blk_in_srcchn_li[n]nth` output of block ID `blk_in_srcid_li[n]`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_srcid_li

Description: Lists block IDs from which the currently scoped block gets its input data. Index this parameter within a loop on `num_blk_in_i` to obtain all the source block ID numbers from which the currently scoped block gets its individual inputs.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_typ_li

Description: Lists the types in integer representation for all the inputs of the currently scoped block. Obtained from the Output Type field of the source block **Outputs** tab. The data types `DOUBLE`, `INTEGER`, `LOGICAL`, `VARDOUBLE`, `VARINTEGER`, `INTEGERTRUNC`, `UNSIGNED FIXED`, `FIXED`, and `UDT` are assigned 1, 2, 3, 4, 5, 6, 7, 8, and 9, respectively. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_in_i`. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_typ_ls

Description: Lists the types for all the inputs of the currently scoped block. Obtained from the Output Type field of the source block **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_typ_name_ls

Description: Lists the type names for all the inputs of the currently scoped block. Obtained from the Output Type field of the source block **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_unit_ls

Description: Lists the unit of measures for all the inputs of the currently scoped block. Obtained from the Output Unit field of the source block **Document** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_in_wsize_li

Description: Lists the word sizes for all the inputs of the currently scoped block. Obtained from the Word Size field of the source block **Document** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_in_i`.

Size: `num_blk_in_i`

Scope Class: Block

Category: DocumentIt

blk_instancename_s

Description: The name of the instance of a SuperBlock.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_keyname_s

Description: The currently scoped block type in a shortened intercap format.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_name_s

Description: The name of the currently scoped block or that appearing in the block reference name of SuperBlock or DataStore. Obtained from the Name field of the block, SuperBlock, STD, or **DataStore Properties** dialog box. If no name was specified, this will be blank.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_out_accu_lr

Description: Lists the accuracies or precisions for all the outputs of the currently scoped block. Obtained from the Output Accuracy field of the **Document** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_accu_ls

Description: Lists the accuracies or precisions for all the outputs of the currently scoped block. Obtained from the Output Accuracy field of the **Document** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_dsc_ls

Description: Lists the textual descriptions for all the outputs of the currently scoped block. Obtained from the Output Comment field of the **Document** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_id_ls

Description: Lists the IDs for all the outputs of the currently scoped block. Obtained from the Output Label field of the **Document** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_maxv_lr

Description: Lists the maximum values for all the outputs of the currently scoped block. Obtained from the Output Maximum field of the **Document** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_maxv_ls

Description: Lists the maximum values for all the outputs of the currently scoped block. Obtained from the Output Maximum field of the **Document** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_memaddr_ls

Description: Lists the memory address values for all the outputs of the currently scoped block. Obtained from the Output Address field of the **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_minv_lr

Description: Lists the minimum values for all the outputs of the currently scoped block. Obtained from the Output Minimum field of the **Document** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_minv_ls

Description: Lists the minimum values for all the outputs of the currently scoped block. Obtained from the Output Minimum field of the **Document** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_mnem_ls

Description: Lists the mnemonic names for all the outputs of the currently scoped block. Obtained from the Mnemonic field of the **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_name_ls

Description: Lists the names for all the outputs of the currently scoped block. Obtained from the Output Name field of the **Outputs** tab. All output names for a block can be retrieved by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_radix_li

Description: Lists the radix numbers for all the outputs of the currently scoped block. Obtained from the Radix field of the **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Note Before using `blk_out_radix_li`, make sure that `blk_out_typ_li` is a fixed-point type. `blk_out_radix_li` is not valid for non-fixed-point types.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_scope_ls

Description: Lists the scope for all the outputs of the currently scoped block. Obtained from the Output Scope field of the **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`. The return value is either `LOCAL` for local scope (a local signal) or `GLOBAL` for global scope (a global signal).

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_typ_li

Description: Lists the types in integer representation for all the outputs of the currently scoped block. Obtained from the Output Type field of the **Outputs** tab. The data types `DOUBLE`, `INTEGER`, `LOGICAL`, `VARDOUBLE`, `VARINTEGER`, `INTEGERTRUNC`, `UNSIGNED FIXED`, `FIXED`, and `UDT` are assigned 1, 2, 3, 4, 5, 6, 7, 8, and 9, respectively. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_typ_ls

Description: Lists the data types for all the outputs of the currently scoped block. Obtained from the Output Type field of the **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_typ_name_ls

Description: Lists the type names for all the outputs of the currently scoped block. Obtained from the Output User Type field of the **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_unit_ls

Description: Lists the units of measure for all the outputs of the currently scoped block. Obtained from the Output Unit field of the **Document** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_out_wsize_li

Description: Lists the word sizes for all the outputs of the currently scoped block. Obtained from the Word Size field of the **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_out_i`.

Size: `num_blk_out_i`

Scope Class: Block

Category: DocumentIt

blk_par_enbl_lb

Description: Lists the whether the block parameters have been enabled for the currently scoped block.

Size: `num_blk_flds_i`

Scope Class: Block

Category: DocumentIt

blk_par_val_ls

Description: Lists the block form field data for all the fields of the currently scoped block.

Size: `num_blk_flds_i`

Scope Class: Block

Category: DocumentIt

blk_par_vars_ls

Description: Lists the block form field data for all the %var fields that are defined in the Xmath stack for the currently scoped block. Retrieve all values for a block by indexing this parameter with `num_blk_parvars_i`.

Size: `num_blk_parvars_i`

Scope Class: Block

Category: DocumentIt

blk_par_vars_s

Description: The names of parameter variables for a block. Block form fields differ, depending upon the type of block. Some of the fields within certain blocks are allowed to contain %var parameters. For form fields containing a %var, this parameter produces the field prompt character string, and the single data value which must be supplied by the user in SystemBuild, separated by a colon. For information regarding %var parameters, refer to the *SystemBuild User Guide*.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_par_vars_sysidx_li

Description: Functions as a cross-reference into the system-scoped tokens `@percentvars_*@`. If a given %var name appears as the *k*th entry of `@blk_par_vars_ls@`, then the *k*th entry of `@blk_par_vars_sysidx_li@` will be the index `j` such that the *j*th entry of `@percentvars_*@` contains information about the %variable in question.

The following example shows how to access AutoCode %var tokens using DocumentIt token data:

```
@LOOPP i=0, i lt num_blk_parvars_i, i = i plus 1@
@   j=blk_par_vars_sysidx_li[i]@
@**@%var=@percentvars_ls[j]@
@ENDLOOPP@
```

Size: `num_blk_parvars_i`

Scope Class: Block

Category: DocumentIt

blk_par_vars_val_ls

Description: Lists the block form field data for all the %var fields of the currently scoped block. This token holds the Xmath run-time value of the %var. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_parvars_i`. By default, it has the default value of the %var.

Size: `num_blk_parvars_i`

Scope Class: Block

Category: DocumentIt

blk_parent_id_i

Description: The ID of the parent SuperBlock of the currently scoped block object.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_state_cmt_ls

Description: Lists the textual descriptions for all the states of the currently scoped block. Obtained from the State Comment field of the **States** tab (for example, from a TimeDelay Block dialog box). All comments for a block can be retrieved by indexing this parameter within a loop on `num_blk_state_i`.

Size: `num_blk_state_i`

Scope Class: Block

Category: DocumentIt

blk_state_name_ls

Description: Lists the names for all the states of the currently scoped block. Obtained from the State Name field of the **States** tab (for example, from a TimeDelay Block dialog box). All names for a block can be retrieved by indexing this parameter within a loop on `num_blk_state_i`.

Size: `num_blk_state_i`

Scope Class: Block

Category: DocumentIt

blk_state_typ_li

Description: Lists the types in integer representation for all the states of the currently scoped block. Obtained from the Output Type field of the **Outputs** tab. The data types `DOUBLE`, `INTEGER`, `LOGICAL`, `VARDOUBLE`, `VARINTEGER`, `INTEGERTRUNC`, `UNSIGNED FIXED`, `FIXED`, and `UDT` are assigned 1, 2, 3, 4, 5, 6, 7, 8, and 9, respectively. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_state_i`. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.

Size: `num_blk_state_i`

Scope Class: Block

Category: DocumentIt

blk_state_typ_ls

Description: Lists the data types for all the states of the currently scoped block. Obtained from the Output Type field of the **Outputs** tab. Retrieve all values for a block by indexing this parameter within a loop on `num_blk_state_i`.

Size: `num_blk_state_i`

Scope Class: Block

Category: DocumentIt

blk_typ_s

Description: The type of currently scoped block as defined by SystemBuild; for example, a gain block.

Size: scalar

Scope Class: Block

Category: DocumentIt

blk_usr_par_ext_ls

Description: Lists the extensions associated with each user parameter in the SuperBlock.

Size: `num_blk_usr_par_i`

Scope Class: Block

Category: DocumentIt

blk_usr_par_ls

Description: Lists the user parameter names as defined in the comment section of the currently scoped block. All user parameter names can be retrieved by indexing this parameter within a loop on `num_blk_usr_par_i`.

Size: `num_blk_usr_par_i`

Scope Class: Block

Category: DocumentIt

busy_ds_registers_i

Description: The total number of DataStore registers needing an access guard because of multiple writers to the same register.

Size: scalar

Scope Class: System

Category: AutoCode

call_cont_init( )

Description: Calls initialization of states function.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

call_cont_subsys( )

Description: Calls continuous subsystem update equations function (C only).

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

childprocs_ins_off_li

Description: Offsets of procedures followed by the tasks to the beginning of their children procedure IDs in `childprocs_ins_li`.

Size: `nprocedures_i + ntasks_i`

Scope Class: System

Category: AutoCode

childprocs_ins_li

Description: Lists the IDs of children procedure instances of procedures followed by the tasks.

Size: sum of all elements of `nchildprocs_ins_li`

Scope Class: System

Category: AutoCode

childprocs_off_li

Description: Lists the offsets of unique procedures followed by the tasks to the beginning of their unique children IDs in `childprocs_li`. For example:

```
@SEGMENT find_procs() INT i,m@
@IFF nprocedures_i gt 0@@
@*
 find all of the unique procedures within the top-level
subsystem
*@
Procedure Name                    #in    #out
-----------------------------------------
@  m = childprocs_off_li[nprocedures_i]@@
@  LOOPP i=0, i lt nchildprocs_li[nprocedures_i],
i=i plus 1@@
@      SCOPE PROCEDURE childprocs_li[m plus i]@@
@*   *@@procedurename_s@@34@@
@      IFF utype_b@@
@         utype_nmembers_i@@40@@
@      ELSE@@
@*      *@ 0@40@@
@      ENDIFF@@
@      IF ytype_b@@
@         ytype_nmembers_i@
@      ELSE@@
@*       *@ 0
@      ENDIFF@@
@  ENDLOOPP@@
@ENDIFF@@
@ENDSEGMENT@
```

Size: `nprocedures_i + ntasks_i`

Scope Class: System

Category: AutoCode

childprocs_li

Description: Lists the unique IDs of children procedures of all the procedures followed by the tasks. For an example, refer to the *childprocs_li* section.

Size: sum of all the elements of `nchildprocs_li`

Scope Class: System

Category: AutoCode

collect_fxpdata( )

Description: Collects the fixed-point operator and conversion data for all of the previously generated subsystems and procedures.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

continuous_b

Description: True if a continuous task is present in the model.

Size: scalar

Scope Class: System

Category: AutoCode

continuous_id_i

Description: Task ID of the continuous task if present.

Size: scalar

Scope Class: System

Category: AutoCode

currentdate_s

Description: Date/time at the time of code or document generation.
Size: scalar
Scope Class: System
Category: AutoCode, DocumentIt

currentfname_pfix_s

Description: Prefix of the current output filename. This assumes value dynamically as when `FILEOPEN` is executed.
Size: scalar
Scope Class: System
Category: AutoCode, DocumentIt

currentfname_s

Description: Current output filename. This assumes value dynamically as when `FILEOPEN` is executed.
Size: scalar
Scope Class: System
Category: AutoCode, DocumentIt

dac_createdate_s

Description: Creation date/time of the current `.dac` (compiled template) file.
Size: scalar
Scope Class: System
Category: AutoCode, DocumentIt

dac_fname_s

Description: `.dac` filename specified in the command syntax.
Size: scalar
Scope Class: System
Category: AutoCode, DocumentIt

declare_cont_info( )

Description: Declares continuous subsystem information structure (C only).
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

declare_cont_init( )

Description: Declares the function which initializes the states (C only).
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

declare_cont_states( )

Description: Declares continuous subsystem state structure (C only).
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

declare_dczeros( )

Description: Generates the needed structure if disconnected signals are used.
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

declare_globals( )

Description: Prints global declarations.
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

declare_percentvars( )

Description: Code to declare percent variables (%vars for C only).
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

declare_procs( )

Description: Prints the currently scoped procedure declarations.
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

declare_structs( )

Description: Prints structure declarations. In multiprocessor, it declares structures/records that are only relevant to the current processor scope.
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

declare_subsystem( )

Description: Prints portions of currently scoped subsystem package declaration (Ada only). In multiprocessor, it allocates the input and output records and their pointers that are only relevant in the current processor scope.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

declare_varblocks( )

Description: Prints code to declare variable blocks (C only).

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

define_cont_info( )

Description: Defines continuous subsystem information structure (C only).

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

define_cont_init( )

Description: Defines the function that initializes the states (C only).

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

define_cont_states( )

Description: Defines continuous subsystem state structure (C only).
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

define_dczeros( )

Description: Generates the needed structure if disconnected signals are used.
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

define_percentvars( )

Description: Prints code to define percent variables (%vars).
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

define_procs( )

Description: Prints the currently scoped procedure definition code.
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

define_structs( )

Description: Prints structure object definitions. In multiprocessor, it defines structures/records that are only relevant to the current processor scope.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

define_subsystem( )

Description: Prints currently scoped subsystem definition code.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

define_varblocks( )

Description: Code to define variable blocks objects.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

desprocs_off_li

Description: Offsets of procedures to the beginning of their unique descendant procedure IDs in the `desprocs_li` file.

Size: sum of all elements of `nchildprocs_li`

Scope Class: System

Category: AutoCode

desprocs_li

Description: IDs of unique descendant procedures of all the unique procedures in the model. The descendant procedures are ordered by their dependencies.

Size: sum of all the elements of `ndesprocs_li`

Scope Class: System

Category: AutoCode

dispatch_subsys( )

Description: Prints code to dispatch tasks (C only). Generates code that dispatches tasks in the current processor scope.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

doublebuf_b

Description: True if the task outputs are double buffered. This is done when `ntasks_i` is greater than 1 or when `pseudorate_b` is true. It is also set to true if `nprocessor_i` is more than 1.

Size: scalar

Scope Class: System

Category: AutoCode

ds_cmt_s

Description: Strings from the Comment field of the currently scoped DataStore.

Size: scalar

Scope Class: DataStore

Category: DocumentIt

ds_cmt_ext_s

Description: The extension associated with the DataStore Comment field.

Size: scalar

Scope Class: Block

Category: DocumentIt

ds_code_cmt_s

Description: If this token is used in the Comment field, comments are automatically pulled out during code generation. For information on how to use this token, refer to the *User-Defined Code Comments* section of Chapter 8, *Customizing AutoCode and Generated Code*, of the *AutoCode User Guide*.

Size: scalar

Scope Class: DataStore

Category: DocumentIt

ds_id_i

Description: The ID of this DataStore, that is, the currently scoped DataStore object.

Size: scalar

Scope Class: DataStore

Category: AutoCode, DocumentIt

ds_name_s

Description: The name of the currently scoped DataStore object.

Size: scalar

Scope Class: DataStore

Category: DocumentIt

ds_names_ls

Description: Lists all DataStores in the model.

Size: scalar

Scope Class: System

Category: DocumentIt

ds_reg_accu_lr

Description: Lists the accuracy or precisions of the registers for the currently scoped DataStore. All individual accuracy or precisions can be retrieved by indexing this parameter within a loop on the `num_reg_in_ds_i` parameter.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_accu_ls

Description: This token is the same as `ds_reg_accu_lr` except that actual representations of the accuracies are printed if the data type is Integer or Logical.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_dsc_ls

Description: Lists the textual descriptions of the registers for the currently scoped DataStore. All individual descriptions can be retrieved by indexing this parameter within a loop on the `num_reg_in_ds_i` parameter.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_initv_lr

Description: Lists the initial values of the registers for the currently scoped DataStore. All individual initial values can be retrieved by indexing this parameter within a loop on the `num_reg_in_ds_i` parameter.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_initv_ls

Description: This token is the same as `ds_reg_initv_lr` except that actual representations of the initial values are printed if the data type is Integer or Logical.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_lbl_ls

Description: Lists the labels of the registers for the currently scoped DataStore. All individual labels can be retrieved by indexing this parameter within a loop on the `num_reg_in_ds_i` parameter.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_maxv_lr

Description: Lists the maximum values of the registers for the currently scoped DataStore. All individual maximum values can be retrieved by indexing this parameter within a loop on the `num_reg_in_ds_i` parameter.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_maxv_ls

Description: This token is the same as `ds_reg_maxv_lr` except that the actual maximum values are printed if the data type is Integer or Logical.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_minv_lr

Description: Lists the minimum values of the registers for the currently scoped DataStore. All individual minimum values can be retrieved by indexing this parameter within a loop on the `num_reg_in_ds_i` parameter.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_minv_ls

Description: This token is the same as `ds_reg_minv_lr` except that each minimum value is a string.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_mnem_ls

Description: Lists the mnemonics of the registers for the currently scoped DataStore. All individual mnemonic names can be retrieved by indexing this parameter within a loop on the `num_reg_in_ds_i` parameter.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_name_ls

Description: Lists all the names of the registers in the currently scoped DataStore.

Size: `num_ds_in_i` or `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_radix_li

Description: Lists the radix numbers of the registers for the currently scoped DataStore. All individual radix numbers can be retrieved by indexing this parameter within a loop on the `num_reg_in_ds_i` parameter.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_typ_li

Description: The DataStore register types in integer representation. The data types `DOUBLE`, `INTEGER`, `LOGICAL`, `VARDOUBLE`, `VARINTEGER`, `INTEGERTRUNC`, `UNSIGNED FIXED`, `FIXED`, and `UDT` are assigned 1, 2, 3, 4, 5, 6, 7, 8, and 9, respectively. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_typ_ls

Description: Lists data types of the registers for the currently scoped DataStore. All individual data types can be retrieved by indexing this parameter within a loop on the `num_reg_in_ds_i` parameter.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_typ_name_ls

Description: Lists the type names of the registers for the currently scoped DataStore. All individual type names can be retrieved by indexing this parameter within a loop on the `num_reg_in_ds_i` parameter.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_unit_ls

Description: Lists the unit of measures of the registers for the currently scoped DataStore. All individual units of measure can be retrieved by indexing this parameter within a loop on the `num_reg_in_ds_i` parameter.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_reg_wsize_li

Description: Lists the word sizes of the registers for the currently scoped DataStore. All individual word sizes can be retrieved by indexing this parameter within a loop on the `num_reg_in_ds_i` parameter.

Size: `num_reg_in_ds_i`

Scope Class: DataStore

Category: DocumentIt

ds_usr_par_ext_ls

Description: Lists extensions associated with each user in the DataStore.

Size: `num_ds_usr_par_i`

Scope Class: DataStore

Category: DocumentIt

ds_usr_par_ls

Description: Lists all the user parameter names as defined in the comment section of the currently scoped DataStore. All individual user parameters can be retrieved by indexing this parameter within a loop on `num_ds_usr_par_i`.

Size: `num_ds_usr_par_i`

Scope Class: DataStore

Category: DocumentIt

dstype_b

Description: True if this DataStore has a structure/record declaration.

Size: scalar

Scope Class: DataStore

Category: AutoCode

dstype_members_ls

Description: Lists member names of this DataStore structure/record.

Size: `dstype_nmembers_i`

Scope Class: DataStore

Category: AutoCode

dstype_nmembers_i

Description: Number of members within this DataStore structure/record.

Size: scalar

Scope Class: DataStore

Category: AutoCode

dstype_obj_s

Description: The name of the object denoting this DataStore definition.
Size: scalar
Scope Class: DataStore
Category: AutoCode

dstype_tag_s

Description: The name of this DataStore structure tag/record.
Size: scalar
Scope Class: DataStore
Category: AutoCode

envflags_i

Description: Value of internal environment flags for the currently scoped subsystem or procedure.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

epsilon_r

Description: Value of `EPSILON`.
Size: scalar
Scope Class: System
Category: AutoCode

errorcheck_b

Description: Indicates if error checking is to be performed after the call to the procedure is made.

Size: scalar

Scope Class: Procedure

Category: AutoCode

errorcheck_option_b

Description: Indicates if the `-e` option has been specified.

Size: scalar

Scope Class: System

Category: AutoCode

extended_procedure_info_b

Description: Indicates if the `-epi` option has been specified.

Size: scalar

Scope Class: System

Category: AutoCode

fast_scheduler_b

Description: Indicates if the faster c-delay scheduler is to be used (`-sched 1`).

Size: scalar

Scope Class: System

Category: AutoCode

fixed_signed_byte_type_i

Description: RT_SBYTE*xx* data type for a %var or variable block variable.
Size: scalar
Scope Class: System
Category: AutoCode

fixed_signed_long_type_i

Description: RT_SLONG*xx* data type for a %var or variable block variable.
Size: scalar
Scope Class: System
Category: AutoCode

fixed_signed_short_type_i

Description: RT_SSHORT*xx* data type for a %var or variable block variable.
Size: scalar
Scope Class: System
Category: AutoCode

fixed_unsigned_byte_type_i

Description: RT_UBYTE*xx* data type for a %var or variable block variable.
Size: scalar
Scope Class: System
Category: AutoCode

fixed_unsigned_long_type_i

Description: RT_ULONG*xx* data type for a %var or variable block variable.
Size: scalar
Scope Class: System
Category: AutoCode

fixed_unsigned_short_type_i

Description: `RT_USHORTxx` data type for a %var or variable block variable.
Size: scalar
Scope Class: System
Category: AutoCode

fixpt_math_b

Description: Indicates if fixed-point math is used in the model.
Size: scalar
Scope Class: System
Category: AutoCode

float_type_i

Description: `RT_FLOAT` data type for a %var or variable block variable.
Size: scalar
Scope Class: System
Category: AutoCode

full_info_b

Description: Indicates if the procedure requires a complete Info structure.
Size: scalar
Scope Class: Procedure
Category: AutoCode

full_uy_required_b

Description: Indicates if structures (U and Y) are to be used for the interface of Procedure SuperBlocks.

Size: scalar

Scope Class: System

Category: AutoCode

fxp_argtype_li

Description: A list containing `FXPPACKEDID` values representing the data types of the left and right arguments for the operator. Unary operators only use the right type, except the ABS operator uses the left type (refer to the *Special Type Encodings* section). Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.

Size: scalar

Scope Class: System

Category: AutoCode

fxp_conversionid_li

Description: A list containing values that represent the type of conversion. The value is used as an index for `fxp_conversions_ls` parameter. The value has the range `include_img` is a TPL function that is a superset of `small_frame` and `large_frame`. This function automatically imports an Enhanced Metafile file named `@sb_mangled_name_s@.emf` (where `sb_mangled_name_s` is a currently scoped SuperBlock name) depending on the scope where the TPL function is invoked. Verify the `.emf` file is generated beforehand using the **Hardcopy SuperBlock(s)** option under the **Build** pull-down menu. Since this function has only a symbolic link to the Enhanced Metafile file, the latest `.emf` file is loaded automatically.

tof –1, 0..17, 1000..1017. A value greater than or equal to 1000 represents a duplicated conversion function. This means that at some other index in this list, the same conversion exists. To obtain which conversion is duplicated, subtract 1000. A value of –1 is not a fixed-point conversion and must not be generated.

Size: scalar

Scope Class: System

Category: AutoCode

fxp_conversions_ls

Description: Provides an ordered list of the names of the generic functions found in the `SA_FIXED_GENERICS` package specification (refer to Table 3-2). These functions are used to instantiate conversion functions.

Size: scalar

Scope Class: System

Category: AutoCode

Table 3-2. Content of fxp_conversions_ls Parameter

Index	Content
0	`INTCAST`
1	`LONGINTCAST`
2	`BOOLEANCAST`
3	Reserved for a fixed to `RT_FLOAT` conversion. This conversion is never instantiated because the language conversion is used.
5	`LONGINTFIXEDCAST`
6	`BOOLEANFIXEDCAST`
7	`FLOATFIXEDCAST`
8	`FIXED_CAST`
9	`NOOPCAST`
10	`INTCAST_TRUNC`
11	`LONGINTCAST_TRUNC`
12	`FLOATFIXEDCAST_TRUNC`
14	`INTCAST_ROUND`
15	`FLOATFIXEDCAST_ROUND`

Table 3-2. Content of fxp_conversions_ls Parameter (Continued)

Index	Content
16	FLOATFIXEDCAST_ROUND
17	FIXED_CAST_ROUND

fxp_convertarg_li

Description: Lists FXPPACKEDID values that represent the fixed-point type(s) used in the conversion. This type is in the right type. Most conversions use only one fixed-point type, either as the type of the argument or the type of the result.

Some conversions require two fixed-point types. In those cases, the right type is the argument type and the left type is the result type (refer to the *Special Type Encodings* section).

Size: scalar

Scope Class: System

Category: AutoCode

fxp_long_fxpname_ls

Description: This parameter contains a list of prefixes of the fixed-point type names. The names are mapped to a special encoding compatible with the code generator's representation of the types. Refer to Table 3-3.

Size: scalar

Scope Class: System

Category: AutoCode

Table 3-3. Content of fxp_long_fxpname_ls

Index	Content
0	—
1	—
2	RT_UBYTE
3	RT_SBYTE

Table 3-3. Content of fxp_long_fxpname_ls (Continued)

Index	Content
4	RT_USHORT
5	RT_SSHORT
6	—
7	—
8	RT_ULONG
9	RT_SLONG
10..33	LONGINTCAST_TRUNC
34	RT_UBYTEn
35	RT_USBYTEn
36	RT_USHORTn
37	RT_SSHORTn
38	—
39	—
40	RT_ULONGn
41	RT_SLONGn

fxp_operatorid_li

Description: Lists the operators to overload. A list containing indices for use with the `fxp_operator_list_ls` and `fxp_operator_name_ls` parameters. Values of the index range from –1, 0..10, 1000..1010. An index greater than or equal to 1000 represents that the operator is a duplicate, meaning that at some other index in the list, the same operator with left, right, and result type information exists. To obtain which operator is a duplicate, subtract 1000 from the number. An index with the value of –1 is not a fixed-point operator and must be skipped.

Size: scalar

Scope Class: System

Category: AutoCode

fxp_operator_list_ls

Description: This parameter provides an ordered list of the names of the Ada operators that can be overloaded. Refer to Table 3-4.

Size: scalar

Scope Class: System

Category: AutoCode

Table 3-4. Content of fxp_operator_list_ls Parameter

Index	Content
0	INTCAST
0	+
1	-
2	*
3	/
4*	-
5	=
6	<
7	<=
8	>
9	>=
9	NOOPCAST
10	ABS
* This index represents unary minus. Unary plus (identity) is not supported.	

fxp_operator_name_ls

Description: This parameter provides an ordered list of the names of the generic functions found in the SA_FIXED_GENERICS package specification. These functions instantiate overloaded operators. Refer to Table 3-5.

Size: scalar

Scope Class: System

Category: AutoCode

Table 3-5. Content of fxp_operator_name_ls Parameter

Index	Content
0	FIXED_ADD
1	FIXED_SUB
2	FIXED_MUL
3	FIXED_DIV
4	NEGATION
5	EQUAL
6	LESSTHAN
7	LESSEQUAL
8	GREATERTHAN
9	GREATEREQUAL
10	ABS

fxp_resulttype_li

Description: A list containing the FXPPACKEDID values representing the result type of the operator. The result data type is in the left packed ID. The right packed ID is reserved for an intermediate fixed-point type. Currently, only addition and subtraction operators use the intermediate type (refer to the *Special Type Encodings* section).

Size: scalar

Scope Class: System

Category: AutoCode

fxp_short_fxpname_ls

Description: This parameter contains a list of abbreviated prefixes for the fixed-point type names. The names are mapped to a special encoding compatible with the code generator's representation of the types. Refer to Table 3-6.

Size: scalar

Scope Class: System

Category: AutoCode

Table 3-6. Content of fxp_conversions_ls Parameter

Index	Content
0	—
1	—
2	UB
3	SB
4	US
5	SS
6	—
7	—
8	UL
9	SL
10..33	—
34	UBn
35	SBn
36	USn
37	SSn
38	—
39	—
40	ULn
41	SLn

get_pvar_init(pvaridx {, validx})

Description: Returns a value in `returnvalue_s` that is the string image of one initial value of the specified %var. The `validx` is optional. When not present, all initial values are code generated and nothing is returned.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

get_varblk_init(varblkidx {, validx})

Description: Returns a value in `returnvalue_s` that is the string image of one initial value of the specified variable block. The `validx` argument is optional and when not present causes all initial values are code generated and nothing is returned. The following example accesses variable block initial values:

```
@LOOPP i=0, i lt nvars_i, i = i plus 1@@
@   vars_ls[i]@ = @
@   j=1@@
@   LOOPP k=0, k lt vars_typ_sfix_dim_li[i],
k = k plus 1@@
@     m = k plus vars_sfix_dim_start_li[i]@@
@     j = j times vars_typ_sfix_li[m]@@
@   ENDLOOPP@@
@
@   LOOPP k=0, k lt j, k = k plus 1@@
@     get_varblk_init(i,k)@@returnvalue_s@,
@
@   ENDLOOPP@
@ENDLOOPP@
```

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

gtype_b

Description: Indicates if the model has subsystem global-scope variables for the model.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

gtype_blk_channel_list_li

Description: Lists the channel (pin) number of the block that writes into the *n*th subsystem global-scope variable. Channel numbers are 0-based.

Size: `gtype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

gtype_blk_list_li

Description: Lists the block number of the block that writes into each subsystem global-scope variable. The block number is a unique number for all blocks in the entire system.

Size: `gtype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

gtype_blk_ordinal_li

Description: Lists the block numbers on which to scope in order to reach the blocks that produced a particular global signal. If the name of a particular global signal is found as the *k*th element of the token `@gtype_members_ls@`, then the block producing this signal can be found by scoping to the block and SuperBlock numbers in `@gtype_blk_ordinal_li@` and `@gtype_sb_ordinal_li@`.

The follwing example shows how to access block information for the global signal:

```
@LOOPP i=0, i lt gtype_nmembers_i, i = i plus 1@@
@**@Global Name: @gtype_members_ls[i]@
@    SCOPE SUPERBLOCK gtype_sb_ordinal_li[i]@@
@**@Superblock Name: @sb_name_s@
@    SCOPE BLOCK gtype_blk_ordinal_li[i]@@
@**@Block Name: @blk_name_s@
@ENDLOOPP@
```

Size: `gtype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

gtype_members_init_lb

Description: Initializes the names of the subsystem global-scope variables.

Size: `gtype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

gtype_members_ls

Description: Lists the names of the subsystem global-scope variables.

Size: `gtype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

gtype_members_memaddr_ls

Description: Lists memory addresses for the subsystem global-scope variables.

Size: `gtype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

gtype_members_size_li

Description: Lists the size of the dimension of the subsystem global-scope variable. A value of 2 or greater indicates the variable is an array.

Size: `gtype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

gtype_members_typ_pfix_ls

Description: Lists the data type name of the subsystem global-scope variables.

Size: `gtype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

gtype_members_typ_sfix_ls

Description: Lists the suffix of the subsystem global-scope variables.

Size: `gtype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

gtype_members_typ_li

Description: Lists the enumerated value of the data type of the subsystem global-scope variables. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.

Size: `gtype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

gtype_members_value_ls

Description: Lists the values of the subsystem global-scope variables.

Size: `gtype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

gtype_nmembers_i

Description: The total number of subsystem global-scope variables.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

gtype_sb_ordinal_li

Description: Contains the SuperBlock numbers on which to scope in order to reach the block that produced a particular global signal. If the name of a particular global signal is found as the *k*th element of the token `@gtype_members_ls@`, then the block producing this signal can be found by scoping to the Block and SuperBlock numbers in `@gtype_blk_ordinal_li@` and `@gtype_sb_ordinal_li@`.

Size: `gtype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

include_img( )

Description: For FrameMaker:

`include_img` is a TPL function that is a superset of `small_frame` and `large_frame`. This function automatically imports an encapsulated PostScript file named `@sb_name_s@.eps` (where `sb_name_s` is a currently scoped SuperBlock name) depending on the scope where the TPL function is invoked. Make sure the `.eps` file is generated beforehand using the **Hardcopy SuperBlock(s)** option under the **Build** pull-down menu. Since this function has only a symbolic link to the encapsulated PostScript file, the latest `.eps` file is loaded automatically.

Note Generate PostScript files from SystemBuild using the **File»Print to File** option; be sure to specify the filename extension `eps`.

The `include_img` function depends on the parameter `@sb_name_s@`, where the filename is `@sb_name_s@.eps` (or `.ps.img`). Verify there are no embedded spaces.

For Microsoft Word:

`include_img` is a TPL function that is a superset of `small_frame` and `large_frame`. This function automatically imports an Enhanced Metafile file named `@sb_mangled_name_s@.emf` (where `sb_mangled_name_s` is a currently scoped SuperBlock name) depending on the scope where the TPL function is invoked. Verify the `.emf` file is generated beforehand using the **Hardcopy SuperBlock(s)** option under the **Build** pull-down menu. Since this function has only a symbolic link to the Enhanced Metafile file, the latest `.emf` file is loaded automatically.

Note Generate PostScript files from SystemBuild using the **File»Print to File** option; be sure to specify the filename extension `emf`.

The `include_img` function depends on parameter `@sb_mangled_name_s@`, where filename is `@sb_mangled_name_s@.emf` (or `.emf.img`). Make sure there are no embedded spaces.

Size: scalar

Scope Class: Function

Category: DocumentIt

initialtaskstate_ls

Description: Initial states of the tasks.
Size: `ntasks_i`
Scope Class: System
Category: AutoCode

initop_b

Description: Indicates if a subsystem has initial outputs or not.
Size: scalar
Scope Class: Subsystem
Category: AutoCode

init_call_required_b

Description: Indicates if the PREINIT phase of a subsytem or INIT phase of a procedure is required.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

inputs_i

Description: Number of external inputs to this subsystem.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

integer_type_i

Description: RT_INTEGER data type for a %var or variable block variable.
Size: scalar
Scope Class: System
Category: AutoCode

integrator_i

Description: Value used to select an integration algorithm for continuous code generation.
Size: scalar
Scope Class: System
Category: AutoCode

interrupt_procs_li

Description: Lists the global number for each interrupt procedure SuperBlock. Use the global number for the argument to SCOPE PROCEDURE.
Size: ninterrupt_procs_i
Scope Class: System
Category: AutoCode

is_blk_b

Description: Determines if the currently scoped block is a block within a SuperBlock.
Size: scalar
Scope Class: Block, STD Block
Category: DocumentIt

is_ds_b

Description: Determines if the currently scoped block is a DataStore.
Size: scalar
Scope Class: Block, STD Block
Category: DocumentIt

is_sb_b

Description: Determines if the currently scoped block is a SuperBlock that appears as a block reference.
Size: scalar
Scope Class: Block, STD Block
Category: DocumentIt

is_std_b

Description: Determines if the currently scoped block is a State Transition Diagram.
Size: scalar
Scope Class: Block, STD Block
Category: DocumentIt

is_text_b

Description: True if the currently scoped block is a text block.
Size: scalar
Scope Class: Block
Category: DocumentIt

itype_b

Description: True if the subsystem/procedure has the info structure/record declaration.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

itype_members_ls

Description: Lists member names of the subsystem/procedure info structure/record.

Size: `itype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

itype_members_size_li

Description: Lists sizes of the dimensions of each member of the subsystem/procedure info structure/record. A value of 2 or greater indicates the variable is an array.

Size: `itype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

itype_members_typ_li

Description: Lists the enumerated value of the data type of the member variables of the subsystem/procedure info structure/record. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.

Size: `itype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

itype_nmembers_i

Description: Number of members within the subsystem/procedure info structure/record.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

itype_tag_s

Description: The name of the subsystem/procedure info type `struct` tag or record.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

kroption_b

Description: Indicates if old-style C (Kernighan and Ritchie) function prototypes are being generated.

Size: scalar

Scope Class: System

Category: AutoCode

language_s

Description: String denoting language of current session (C or Ada).

Size: scalar

Scope Class: System

Category: AutoCode

large_frame( )

Description: A TPL function that can be used to generate an anchored frame of size 7 × 5 in. for FrameMaker target documents. An empty box appears when you invoke the document. You can import `.eps` images into the anchored frame using the Capture feature in the desktop publishing software application. Refer to the *small_frame( )* section.

Size: scalar

Scope Class: System

Category: AutoCode

level_i

Description: The level of the currently scoped SuperBlock (0 = top).

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

logical_type_i

Description: `RT_BOOLEAN` data type for a %var or variable block variable.

Size: scalar

Scope Class: System

Category: AutoCode

macro_procs_ls

Description: Lists the names of all the macro procedure SuperBlocks used in the mode.

Size: `nmacro_procs_i`

Scope Class: System

Category: AutoCode

mealy_out_dsc_ls

Description: Lists the Mealy mode outputs of the transitions for each of the bubbles in the currently scoped STD block. All values for the STD block can be retrieved by indexing this parameter within a loop on `num_bbls_in_std_i` while offsetting the index by using `bbl_trn_off_li`. For information on Mealy mode outputs, refer to the *State Transition Diagram Block User Guide*.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

modelname_s

Description: Prefix of the `rtf` filename without the extension.

Size: scalar

Scope Class: System

Category: AutoCode, DocumentIt

moore_out_dsc_ls

Description: Lists the Moore mode outputs for each of the bubbles in the currently scoped STD block. All values for the STD block can be retrieved by indexing this parameter within a loop on `num_bbls_in_std_i`. For information on Moore mode outputs, refer to the *State Transition Diagram Block User Guide*.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

multiprocessor_b

Description: True if number of processors is >1.

Size: scalar

Scope Class: System

Category: AutoCode

n_user_defined_type_i

Description: The number of user-defined types used in the model.

Size: scalar

Scope Class: System

Category: AutoCode

nbackgnd_procs_i

Description: Number of background procedure SuperBlocks.

Size: scalar

Scope Class: System

Category: AutoCode

nchildprocs_ins_li

Description: Lists the number of children procedure instances for procedures followed by tasks. The numbers of children for the procedures are stored first, followed by the numbers of children for the tasks.

Size: `nprocedures_i + ntasks_i`

Scope Class: System

Category: AutoCode

nchildprocs_li

Description: Number of unique children procedures of procedures first and then the tasks. The numbers of unique children for the procedures are stored first, followed by the numbers of children for the tasks.

Size: `nprocedures_i + ntasks_i`

Scope Class: System

Category: AutoCode

nds_i

Description: Number of DataStores in the model.

Size: scalar

Scope Class: System

Category: AutoCode, DocumentIt

ndesprocs_li

Description: Number of all the unique descendant procedures ordered by their dependencies for all unique procedures.

Size: `nprocedures_i`

Scope Class: System

Category: AutoCode

nenabled_i

Description: Number of enabled tasks.

Size: scalar

Scope Class: System

Category: AutoCode

nfxp_conversions_i

Description: Number of conversion functions to generate. The *n*th element of this list parameter and `sp_fxp_conversions_i` represent data pertaining to the *n*th conversion function that is to be generated.

Size: scalar

Scope Class: System

Category: AutoCode

nfxp_opfunctions_i

Description: Number of operator overloaded functions to generate. The *n*th element of this list parameter and `sp_fxp_opfunctions_i` represent data pertaining to the *n*th overloaded function that is to be generated.

Size: scalar

Scope Class: System

Category: AutoCode

ninterrupt_procs_i

Description: Number of interrupt procedure SuperBlocks.

Size: scalar

Scope Class: System

Category: AutoCode

nmacro_procs_i

Description: Total number of macro procedure SuperBlocks used in the model.

Size: scalar

Scope Class: System

Category: AutoCode

nobusmap_b

Description: True if the `-nomap` option is given to turn off the map structure.

Size: scalar

Scope Class: System

Category: AutoCode

noerr_option_b

Description: Indicates if `-noerr` option has been specified.
Size: scalar
Scope Class: System
Category: AutoCode

noicmap_b

Description: Indicates if `-noicmap` option is given to turn off initial condition remap (`XREMAP`).
Size: scalar
Scope Class: System
Category: AutoCode

norestart_b

Description: Indicates if `-Onorestart` option has been specified.
Size: scalar
Scope Class: System
Category: AutoCode

npercentvars_i

Description: Number of %vars in the model.
Size: scalar
Scope Class: System
Category: AutoCode

npercentvars_used_i

Description: Number of %vars used within the currently scoped subsystem/procedure.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

nperiodic_i

Description: Number of periodic tasks.
Size: scalar
Scope Class: System
Category: AutoCode

nprocedures_i

Description: Number of unique procedures in the model.
Size: scalar
Scope Class: System
Category: AutoCode

nprocessors_i

Description: Number of processors in the system.
Size: scalar
Scope Class: System
Category: AutoCode

nprocs_i

Description: Total number of procedures called by the subsystem/procedure.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

nstandard_procs_i

Description: Number of standard procedure SuperBlocks.
Size: scalar
Scope Class: System
Category: AutoCode

nstartup_procs_i

Description: Number of startup procedure SuperBlocks.
Size: scalar
Scope Class: System
Category: AutoCode

nsupblks_i

Description: Number of SuperBlocks in the system.
Size: scalar
Scope Class: System
Category: DocumentIt

ntasks_i

Description: Number of tasks in the system.
Size: scalar
Scope Class: System
Category: AutoCode

ntriggered_i

Description: Number of triggered tasks.
Size: scalar
Scope Class: System
Category: AutoCode

nucbs_i

Description: Number of UCBs used within the scoped subsystem or procedure.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

num_bbl_trn_in_li

Description: Lists the number of bubble transition inputs for each of the bubbles in the currently scoped STD block. All values for the STD block can be retrieved by indexing this parameter within a loop on `num_bbls_in_std_i`.
Size: `num_bbls_in_std_i`
Scope Class: STD Block
Category: DocumentIt

num_bbl_trns_li

Description: Lists the transitions for each of the bubbles in the currently scoped STD block. All values for the STD block can be retrieved by indexing this parameter within a loop on `num_bbls_in_std_i`.
Size: `num_bbls_in_std_i`
Scope Class: STD Block
Category: DocumentIt

num_bbls_in_std_i

Description: The number of bubbles in the currently scoped STD block.

Size: scalar

Scope Class: STD Block

Category: DocumentIt

num_blk_flds_i

Description: The number of fields for the currently scoped block. All basic blocks have fields, but the number of fields for each block differs.

Size: scalar

Scope Class: Block

Category: DocumentIt

num_blk_in_i

Description: The number of inputs for the currently scoped block. Not all blocks have inputs. The Boolean parameter `blk_has_in_b` can be used to determine if a block has inputs.

Size: scalar

Scope Class: Block

Category: DocumentIt

num_blk_out_i

Description: The number of outputs for the currently scoped block. Not all blocks have outputs; the Boolean parameter `blk_has_out_b` can be used to determine if a block has outputs.

Size: scalar

Scope Class: Block

Category: DocumentIt

num_blk_parvars_i

Description: The number of %vars of the block field data defined in the Xmath stack for the currently scoped block.

Size: scalar

Scope Class: Block

Category: DocumentIt

num_blk_state_i

Description: The number of states for the currently scoped block. Not all blocks have states; the Boolean parameter `blk_has_state_b` can be used to determine if a block has states.

Size: scalar

Scope Class: Block

Category: DocumentIt

num_blk_usr_par_i

Description: The number of user-defined parameters in the currently scoped block.

Size: scalar

Scope Class: Block

Category: DocumentIt

num_blks_in_sb_i

Description: The number of blocks in the currently scoped SuperBlock.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

num_ds_in_i

Description: The number of inputs for the currently scoped DataStore.

Size: scalar

Scope Class: DataStore

Category: DocumentIt

num_ds_out_i

Description: The number of outputs for the currently scoped DataStore.

Size: scalar

Scope Class: DataStore

Category: DocumentIt

num_ds_usr_par_i

Description: The number of user-defined parameters in the currently scoped DataStore.

Size: scalar

Scope Class: DataStore

Category: DocumentIt

num_mealy_out_li

Description: Lists the number of Mealy outputs for each of the bubbles in the currently scoped STD block. All values for the STD block can be retrieved by indexing this parameter within a loop on `num_bbls_in_std_i`.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

num_moore_out_li

Description: Lists the number of Moore outputs for each of the bubbles in the currently scoped STD block. All values for the STD block can be retrieved by indexing this parameter within a loop on `num_bbls_in_std_i`.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

num_reg_in_ds_i

Description: The number of registers in the currently scoped DataStore.

Size: scalar

Scope Class: DataStore

Category: DocumentIt

num_sb_in_i

Description: The number of inputs for the currently scoped SuperBlock.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

num_sb_mapped_vars_i

Description: The total number of mapped vars of the currently scoped component.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

num_sb_mapping_vars_i

Description: The total number of mapping vars of the currently scoped component.
Size: scalar
Scope Class: SuperBlock
Category: DocumentIt

num_sb_out_i

Description: The number of outputs for the currently scoped SuperBlock.
Size: scalar
Scope Class: SuperBlock
Category: DocumentIt

num_sb_usr_par_i

Description: The number of user parameters defined in the comment field of the currently scoped SuperBlock.
Size: scalar
Scope Class: SuperBlock
Category: DocumentIt

numip_i

Description: Number of integer parameters.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

numrp_i

Description: Number of real parameters.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

numxs_i

Description: Number of states.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

numin_i

Description: Number of system external inputs.
Size: scalar
Scope Class: System
Category: AutoCode, DocumentIt

numiparsym_i

Description: The total number of integer parameters (IP) used.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

numout_i

Description: Number of system external outputs.
Size: scalar
Scope Class: System
Category: AutoCode, DocumentIt

numrparsym_i

Description: The total number of real parameters (RP) used.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

num_uni_sb_i

Description: Number of unique SuperBlocks in the model.
Size: scalar
Scope Class: System
Category: DocumentIt

nvars_i

Description: Number of variable block variables in the model.
Size: scalar
Scope Class: System
Category: AutoCode

ordered_procs_li

Description: Unique procedure IDs ordered by their dependencies.
Size: `nprocedures_i`
Scope Class: System
Category: AutoCode

outputcount_li

Description: Output counts of the tasks.
Size: `ntasks_i`
Scope Class: System
Category: AutoCode

outputfile_dir_s

Description: Directory prefix of the specified output filename (the main stream). Its value is a `NULL` string if no directory has specification in the output filename.

Size: scalar

Scope Class: System

Category: AutoCode, DocumentIt

outputfile_ext_s

Description: Output filename extension.

Size: scalar

Scope Class: System

Category: AutoCode, DocumentIt

outputfile_pfix_s

Description: Prefix of the output filename of the main stream with the directory portion stripped.

Size: scalar

Scope Class: System

Category: AutoCode, DocumentIt

output_fname_s

Description: Output filename of the main stream. Takes the value from the command syntax if explicitly specified. Otherwise, it takes the name of the `rtf` file with a language-specific suffix command syntax to be generated in the current directory.

Size: scalar

Scope Class: System

Category: AutoCode, DocumentIt

outputs_i

Description: Number of external outputs.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

pe_backgnd_procs_li

Description: Lists the global number for each background procedure SuperBlock for the currently scoped processor. Use the global number for the argument to SCOPE PROCEDURE.

Size: `pe_nbackgnd_procs_i`

Scope Class: Processor

Category: AutoCode

pe_id_i

Description: ID of the currently scoped processor. ID ranges from 0 to `nprocessor_i` – 1.

Size: scalar

Scope Class: Processor

Category: AutoCode

pe_interrupt_procs_li

Description: Lists the global number for each interrupt procedure SuperBlock for the currently scoped processor. Use the global number for the argument to SCOPE PROCEDURE.

Size: `pe_ninterrupt_procs_i`

Scope Class: Processor

Category: AutoCode

pe_macro_procs_ls

Description: Lists the names of macro procedure SuperBlocks for the currently scoped processor.

Size: `pe_nmacro_procs_i`

Scope Class: Processor

Category: AutoCode

pe_nbackgnd_procs_i

Description: Number of background procedure SuperBlocks on currently scoped processor.

Size: scalar

Scope Class: Processor

Category: AutoCode

pe_ninterrupt_procs_i

Description: Number of interrupt procedure SuperBlocks on currently scoped processor.

Size: scalar

Scope Class: Processor

Category: AutoCode

pe_nmacro_procs_i

Description: Number of macro procedure SuperBlocks on currently scoped processor.

Size: scalar

Scope Class: Processor

Category: AutoCode

pe_npercentvars_i

Description: Number of %vars on currently scoped processor.
Size: scalar
Scope Class: Processor
Category: AutoCode

pe_nprocedures_i

Description: Number of procedure SuperBlocks on currently scoped processor.
Size: scalar
Scope Class: Processor
Category: AutoCode

pe_nstandard_procs_i

Description: Number of standard procedure SuperBlocks on currently scoped processor.
Size: scalar
Scope Class: Processor
Category: AutoCode

pe_nstartup_procs_i

Description: Number of startup procedure SuperBlocks on currently scoped processor.
Size: scalar
Scope Class: Processor
Category: AutoCode

pe_ntasks_i

Description: Number of startup procedure SuperBlocks on currently scoped processor.

Size: scalar

Scope Class: Processor

Category: AutoCode

pe_nvars_i

Description: Number of variable block variables on currently scoped processor.

Size: scalar

Scope Class: Processor

Category: AutoCode

pe_ordered_procs_li

Description: Lists the global number for all procedure SuperBlocks for the currently scoped processor. Use the global number for the argument to SCOPE PROCEDURE.

Size: `pe_nprocedures_i`

Scope Class: Processor

Category: AutoCode

pe_percentvar_map_li

Description: Lists the global number for each %var used on the currently scoped processor.

Size: `pe_npercentvars_i`

Scope Class: Processor

Category: AutoCode

pe_standard_procs_li

Description: Lists the global number for each standard procedure SuperBlock for the currently scoped processor. Use the global number for the argument to SCOPE PROCEDURE.

Size: `pe_nstandard_procs_i`

Scope Class: Processor

Category: AutoCode

pe_startup_procs_li

Description: Lists the global number for each startup procedure SuperBlock for the currently scoped processor. Use the global number for the argument to SCOPE PROCEDURE.

Size: `pe_nstartup_procs_i`

Scope Class: Processor

Category: AutoCode

pe_var_map_li

Description: Lists the global number for each variable block variable used on the currently scoped processor.

Size: `pe_nvars_i`

Scope Class: Processor

Category: AutoCode

percentvars_ls

Description: Lists the names of all the %vars in the model. The following example shows values of `percentvars_*` parameters. Assume the following are declarations of %vars within the generated code:

```
RT_INTEGER climb_rate;
RT_FLOAT fuel_mixture[3][4][5];
RT_INTEGER hilow[2][2];
RT_INTEGER position[10];
```

The following are the values of the %var parameters encapsulating the information relevant to those %var variables.

```
percentvars_ls[0] = "climb_rate"
percentvars_ls[1] = "fuel_mixture"
percentvars_ls[2] = "hilow"
percentvars_ls[3] = "position"
```

```
percentvars_typ_pfix_ls[0] = "RT_INTEGER"
percentvars_typ_pfix_ls[1] = "RT_FLOAT"
percentvars_typ_pfix_ls[2] = "RT_INTEGER"
percentvars_typ_pfix_ls[3] = "RT_INTEGER"
```

```
percentvars_typ_sfix_li[0] = 0
percentvars_typ_sfix_li[1] = 3
percentvars_typ_sfix_li[2] = 4
percentvars_typ_sfix_li[3] = 5
percentvars_typ_sfix_li[4] = 2
percentvars_typ_sfix_li[5] = 2
percentvars_typ_sfix_li[6] = 10;
```

```
percentvars_typ_sfix_dim_li[0] = 0
percentvars_typ_sfix_dim_li[1] = 3
percentvars_typ_sfix_dim_li[2] = 2
percentvars_typ_sfix_dim_li[3] = 1
```

```
percentvars_sfix_dim_start_li[0] = 0
percentvars_sfix_dim_start_li[1] = 1
percentvars_sfix_dim_start_li[2] = 4
percentvars_sfix_dim_start_li[3] = 6
```

Size: `npercentvars_i`

Scope Class: System

Category: AutoCode

percentvars_as_varblk_li

Description: Lists values indicating whether the particular %var variable also is used as a variable block within the model. The value is an index to the variable block that is the same as the %var. A value of –1 means that the %var is not a variable block.

Size: `npercentvars_i`

Scope Class: System

Category: AutoCode

percentvars_prsr_scope_li

Description: Lists values indicating the processor scope of each %var variable in the model relative to the last processor scoped upon. A value of 0 indicates not local to current processor and not shared; 1 indicates local to current processor and not shared; 2 indicates not local to processor but shared; and 3 indicates local to processor and shared.

The following example shows the usage of this token where the TPL code is using tokens to generate %var variable declarations in C.

```
@INT offset, k, m@

@LOOPP k=0, k lt npercentvars_i, k=k plus 1@@
@IFF percentvars_prsr_scope_li[k] eq 1 @
   @percentvars_typ_pfix_ls[k]@ @percentvars_ls[k]@@
@offset = percentvars_sfix_dim_start_li[k]@@
@LOOPP m=0, m lt percentvars_typ_sfix_dim_li[k],
m=m plus 1@@
   [@percentvars_typ_sfix_li[offset]@]@
@offset = offset plus 1@@
@ENDLOOPP@@
;
@ENDIFF@@
@ENDLOOPP@@
```

Size: `npercentvars_i`

Scope Class: System

Category: AutoCode

percentvars_sfix_dim_start_li

Description: An index into the start of the `percentvars_typ_sfix_li` for each %var variable. Refer to the *percentvars_ls* section.

Size: `npercentvars_i`

Scope Class: System

Category: AutoCode

percentvars_subtype_li

Description: Subtype information for the %var variables. If the type is non-fixed point, the value is 0. If fixed-point, the value is the radix of the type.

Size: `npercentvars_i`

Scope Class: System

Category: AutoCode

percentvars_typ_pfix_ls

Description: Lists the type specification of each of the %var variables used in the model. Refer to the *percentvars_ls* section.

Size: `npercentvars_i`

Scope Class: System

Category: AutoCode

percentvars_typ_sfix_li

Description: The size of each dimension for each %var variable used in the model. If a variable is scalar, its dimension is considered 0. Refer to the *percentvars_ls* section.

Size: Sum of all dimensions of all %vars. A scalar var counts as one in the list count.

Scope Class: System

Category: AutoCode

percentvars_typ_sfix_dim_li

Description: Lists the number of dimensions for each %var variable used in the model. Scalar variables have a dimension of zero (0). Compare the values against the set of data type tokens (`*_type_i`) to determine the data type. Refer to the *percentvars_ls* section.

Size: `npercentvars_i`

Scope Class: System

Category: AutoCode

percentvars_type_li

Description: An enumerated value representing the type of the %var variable. The elements in the list can be compared to the following constant parameters which represent a specific data type:

`integer_type_i, logical_type_i, fixed_signed_long_i, fixed_unsigned_long_i, fixed_signed_short_i, fixed_unsigned_short_i, fixed_signed_byte_i, fixed_unsigned_byte_i, and user_type_i.`

Refer to the *vars_ls* section for the usage of this list.

Size: `npercentvars_i`

Scope Class: System

Category: AutoCode

percentvars_used_li

Description: Lists the global IDs of all %vars used within the currently scoped subsystem/procedure. This list does not contain the %vars used in any nested procedure(s).

Size: `npercentvars_used_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

procedurename_s

Description: Name of the procedure/subsystem.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

processor_i

Description: Parent processor ID.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

processorid_li

Description: Lists processor IDs corresponding to the tasks. Processor ID ranges from 0 to `nprocessor_i` – 1.
Size: `ntasks_i`
Scope Class: System
Category: AutoCode

proc_id_li

Description: List containing the identification numbers of the procedure SuperBlocks used in scoped subsystem or procedure.
Size: `nprocs_i`
Scope Class: Subsystem/Procedure
Category: AutoCode

proc_info_parameters_b

Description: Indicates if parameter data is present in the procedure's Info structure.
Size: scalar
Scope Class: Procedure
Category: AutoCode

proc_info_percentvars_b

Description: Indicates if %vars are present in the procedure's Info structure.
Size: scalar
Scope Class: Procedure
Category: AutoCode

proc_initneeded_b

Description: Indicates if only the INIT state flag is needed to be passed as a replacement for the INFO structure for the currently scoped procedure. This can only occur if the `-noinfo` option is used and the procedure requires just the INIT state flag.
Size: scalar
Scope Class: Procedure
Category: AutoCode

proc_intr_name_s

Description: Name of interrupt for currently scoped interrupt procedure.
Size: scalar
Scope Class: Procedure
Category: AutoCode

proc_mode_flags_ls

Description: Lists mode flags for all procedures. Access values by the procedure global reference number minus `nstandard_procs_i`. For RTOS information, refer to Chapter 4, *Generating Code for Real-Time Operating Systems*, of the *AutoCode Reference*.

Size: `nprocedures_i`

Scope Class: System

Category: AutoCode

proc_names_ls

Description: Lists the names of the procedure SuperBlocks called from the scoped subsystem or procedure.

Size: `nprocs_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

proc_nsbs_i

Description: Number of SuperBlocks corresponding to this subsystem/procedure.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

proc_ordering_li

Description: Lists orderings used to determine a relative calling order between background and interrupt procedures. Access values by the procedure global reference number minus `nstandard_procs_i`. For RTOS information, refer to Chapter 4, *Generating Code for Real-Time Operating Systems*, of the *AutoCode Reference*.

Size: `nprocedures_i minus nstandard_procs_i`

Scope Class: System

Category: AutoCode

proc_priority_li

Description: Lists priorities assigned to interrupt and background procedure SuperBlocks. Access values by the procedure global reference number minus `nstandard_procs_i`. For RTOS information, refer to Chapter 4, *Generating Code for Real-Time Operating Systems*, of the *AutoCode Reference*.

Size: `nprocedures_i minus nstandard_procs_i`

Scope Class: System

Category: AutoCode

proc_processor_map_li

Description: Lists processor assignments for all non-standard procedure SuperBlocks. Access values by the procedure global reference number minus `nstandard_procs_i`. For RTOS information, refer to Chapter 4, *Generating Code for Real-Time Operating Systems*, of the *AutoCode Reference*.

Size: `nprocedures_i minus nstandard_procs_i`

Scope Class: System

Category: AutoCode

proc_sb_id_li

Description: Lists IDs of SuperBlocks corresponding to this subsystem/procedure.

Size: `proc_nsbs_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

proc_stack_size_li

Description: Lists stack size values for all non-standard procedure SuperBlocks. Access values by the procedure global reference number minus `nstandard_procs_i`. For RTOS information, refer to Chapter 4, *Generating Code for Real-Time Operating Systems*, of the *AutoCode Reference*.

Size: `nprocedures_i minus nstandard_procs_i`

Scope Class: System

Category: AutoCode

proc_ucb_hook( )

Description: Prints UCB wrapper code around the currently scoped reusable procedure.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

proc_userblock_b

Description: True if this subsystem/procedure uses any UserCode Block.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

proc_vector_li

Description: Lists vector values for all interrupt procedure SuperBlocks. Background and Startup procedures do not have a vector value. Access values by the interrupt procedure global reference number minus `nstandard_procs_i`. For RTOS information, refer to Chapter 4, *Generating Code for Real-Time Operating Systems*, of the *AutoCode Reference*.

Size: `nprocedures_i minus nstandard_procs_i`

Scope Class: System

Category: AutoCode

procs_only_b

Description: True if **Procedures Only** option is chosen.

Size: scalar

Scope Class: System

Category: AutoCode

prsr_ip_name_ls

Description: Lists IP names associated with each processor. Refer to the *sspriority_li* section for the usage of this list.

Size: `nprocessors_i`

Scope Class: System

Category: AutoCode

pseudorate_b

Description: This is true if the scheduler frequency is different from that of the fastest task in the system.

Size: scalar

Scope Class: System

Category: AutoCode

pstype_b

Description: True if the subsystem/procedure has the private states structure/record declaration.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

pstype_members_ls

Description: Lists member names of the subsystem/procedure private state structure/record.

Size: `pstype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

pstype_members_size_li

Description: Lists the sizes of the dimension of each member variable of the subsystem/procedure private state structure/record. A value of 2 or greater indicates an array.

Size: `pstype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

pstype_members_typ_li

Description: Lists the enumerated value of the data type of each member variable of the subsystem/procedure private state structure/record. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.

Size: `pstype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

pstype_nmembers_i

Description: Number of members within the subsystem/procedure private state structure/record.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

pstype_tag_s

Description: The name of the subsystem/procedure private state type `struct` tag or record.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

returnvalue_s

Description: Contains the returned string value from the last called TPL function that returns a string.

Size: scalar

Scope Class: System

Category: AutoCode

reset_b

Description: Indicates if the continuous system is resettable or not.

Size: scalar

Scope Class: System

Category: AutoCode

rtf_fname_s

Description: `rtf` filename specified in the command syntax.
Size: scalar
Scope Class: System
Category: AutoCode, DocumentIt

rtos_option_b

Description: Indicates if `-rtos` or `-rtosf` option specified.
Size: scalar
Scope Class: System
Category: AutoCode

sample_hold({subsysId})

Description: Prints code to sample and hold task inputs. In multiprocessor, it generates code that does sample and hold for all the tasks in the current processor scope. If the optional `subsysId` argument is specified, then only the sample and hold code for that task is generated.
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

sb_actv_sig_s

Description: The enabling or triggering signal (number) for the currently scoped SuperBlock. The Trigger Signal field of the SuperBlock **Attributes** tab provides this value for enabled and triggered SuperBlocks, respectively.
Size: scalar
Scope Class: SuperBlock
Category: DocumentIt

sb_attr_s

Description: The SuperBlock attribute type (continuous, discrete, trigger, or procedure) from the SuperBlock Type field of the currently scoped SuperBlock **Attributes** tab.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

sb_cmt_ext_s

Description: The extension associated with the SuperBlock **Comment** tab.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

sb_cmt_s

Description: The contents of the currently scoped SuperBlock **Comment** tab. The content will be unformatted.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

sb_code_cmt_s

Description: A reserved variable that can be declared in the SuperBlock **Comment** tab. The comment associated with this parameter will be pulled out automatically during code generation. In DocumentIt, this variable can be pulled out in SuperBlock scope using `sb_code_cmt_s` instead of the `user-param` function. For information on how to use this token, refer to the *User-Defined Code Comments* section of Chapter 8, *Customizing AutoCode and Generated Code*, of the *AutoCode User Guide*.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_accu_lr

Description: Lists the accuracies or precisions of the currently scoped SuperBlock input channels. These values come from the Input Accuracy field of the SuperBlock **Document** tab. All input accuracies for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_in_i` parameter.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_accu_ls

Description: Lists the accuracies or precisions for all the inputs of the currently scoped SuperBlock. Obtained from the Input Accuracy field of the SuperBlock **Document** tab. All values for a SuperBlock can be retrieved by indexing this parameter within a loop on `num_sb_in_i`.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_dsc_ls

Description: Lists the textual descriptions of the currently scoped SuperBlock input channels. These values come from the Input Label field of the SuperBlock **Document** tab. All input descriptions for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_in_i` parameter.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_id_ls

Description: Lists the IDs for the currently scoped SuperBlock inputs. Each of these comes from the Input Label field of the SuperBlock **Document** tab. All input IDs for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_in_i` parameter.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_maxv_lr

Description: Lists the maximum values of the input data for the channels of the currently scoped SuperBlock. These values come from the Input Max field of the SuperBlock **Document** tab. All input maximum values for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_in_i` parameter.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_maxv_ls

Description: Lists the maximum values for all the inputs of the currently scoped SuperBlock. Obtained from the Input Max field of the SuperBlock **Document** tab. All values for a SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_in_i` parameter.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_minv_lr

Description: Lists the minimum values of the input data for the channels of the currently scoped SuperBlock. These values come from the Input Min field of the SuperBlock **Document** tab. All input minimum values for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_in_i` parameter.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_minv_ls

Description: Lists the minimum values of the input data for the channels of the currently scoped SuperBlock. These values come from the Input Min field of the SuperBlock **Document** tab. All input minimum values for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_in_i` parameter.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_mnem_ls

Description: Lists the mnemonic names for all the inputs of the currently scoped SuperBlock. Obtained from the Mnemonic field of the SuperBlock **Attributes Input** tab. Retrieve all values for a block by indexing this parameter within a loop on the `num_sb_in_i` parameter.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_name_ls

Description: Lists the names for the currently scoped SuperBlock inputs. Each of these comes from the Input Name field of the SuperBlock **Document** tab. All input names for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_in_i` parameter.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_radix_li

Description: Lists the radix numbers for all the inputs of the currently scoped SuperBlock. Obtained from the Input Radix field of the SuperBlock **Inputs** tab. All values for a SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_in_i` parameter.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_scope_ls

Description: Lists the scope classes for all the inputs of the currently scoped SuperBlock. Obtained from the Input Scope field of the SuperBlock **Inputs** tab. Values only valid for Standard Procedure SuperBlocks. A value of LOCAL indicates Local Scope while a value of GLOBAL indicates Global Scope.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_typ_li

Description: Lists the types in integer representation for all the inputs of the currently scoped SuperBlock. Obtained from the Input Type of the SuperBlock **Inputs** tab. The data types `DOUBLE`, `INTEGER`, `LOGICAL`, `VARDOUBLE`, `VARINTEGER`, `INTEGERTRUNC`, `UNSIGNED FIXED`, `FIXED`, and `UDT` are assigned 1, 2, 3, 4, 5, 6, 7, 8, and 9, respectively. All values for a SuperBlock can be retrieved by indexing this parameter within a loop on `num_sb_in_i`. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_typ_ls

Description: Lists the data types of the input channels of the currently scoped SuperBlock. These values come from the Input Type field of the SuperBlock **Inputs** tab. All input types for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_in_i` parameter.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_typ_name_ls

Description: Lists the type names for all the inputs of the currently scoped SuperBlock. Obtained from the Input User Type field of the SuperBlock **Attributes** tab. All values for a SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_in_i` parameter.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_unit_ls

Description: Lists the units of measure of the input channels of the currently scoped SuperBlock. These values come from the Input Unit field of the SuperBlock **Document** tab. All input units for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_in_i` parameter.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extin_wsize_li

Description: Lists the word sizes for all the inputs of the currently scoped SuperBlock. Obtained from the Word Size field of the SuperBlock Attributes **Inputs** tab. Retrieve all values for a block by indexing this parameter within a loop on the `num_sb_in_i` parameter.

Size: `num_sb_in_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_accu_lr

Description: Lists the accuracies or precisions of the currently scoped SuperBlock output channels. These values come from the Output Accuracy field of the SuperBlock Attributes **Document** tab. All output data types for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_accu_ls

Description: Lists the accuracies or precisions for all the outputs of the currently scoped SuperBlock. Obtained from the Output Accuracy field of the SuperBlock **Document** tab. All values for a SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_dsc_ls

Description: Lists the textual descriptions of the currently scoped SuperBlock output channels. These values come from the Output Comment field of the derived block **Document** tab. All output descriptions for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_id_ls

Description: Lists the IDs for the currently scoped SuperBlock outputs. These values come from the Output Label field of the derived block **Document** tab. All output IDs for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_maxv_lr

Description: Lists the maximum values of the output data for the channels of the currently scoped SuperBlock. These values come from the Output Maximum field of the derived block **Document** tab. All output data types for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_maxv_ls

Description: Lists the maximum values for all the outputs of the currently scoped SuperBlock. Obtained from the Output Maximum field of the derived block **Document** tab. All values for a SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_memaddr_ls

Description: Lists the user-entered memory addresses for the currently scoped SuperBlock outputs. Each of these comes from the Output Address field of the derived block **Label** tab. All memory addresses for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter. Only valid for Standard Procedure SuperBlocks.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_minv_lr

Description: Lists the minimum values of the output data for the channels of the currently scoped SuperBlock. These values come from the Output Minimum field of the derived block **Document** tab. All output minimum values for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_mnem_ls

Description: Lists the mnemonic names for all the outputs of the currently scoped SuperBlock. Obtained from the Mnemonic field of the derived block **Outputs** tab. All values for a SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_name_ls

Description: Lists the names for the currently scoped SuperBlock outputs. Each of these comes from the Output Name field of the derived block **Document** tab. All output names for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_radix_li

Description: Lists the radix numbers for all outputs of the currently scoped SuperBlock. Obtained from the Output Radix field of the derived block **Outputs** tab. All values for a SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_scope_ls

Description: Lists the scope names of the output channels of the currently scoped SuperBlock. These values come from the Output Scope field of the derived block **Outputs** tab. A value of LOCAL indicates a Local Scope, while a value of GLOBAL indicates Global Scope.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_typ_li

Description: Lists the types in integer representation for all the outputs of the currently scoped SuperBlock. Obtained from the Output Data Type of the derived block **Outputs** tab. The data types `DOUBLE`, `INTEGER`, `LOGICAL`, `VARDOUBLE`, `VARINTEGER`, `INTEGERTRUNC`, `UNSIGNED FIXED`, `FIXED`, and `UDT` are assigned 1, 2, 3, 4, 5, 6, 7, 8, and 9, respectively. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type. All values for a SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_typ_ls

Description: Lists the data types of the output channels of the currently scoped SuperBlock. These values come from the Output Data Type field of the derived block **Outputs** tab. All output data types for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_typ_name_ls

Description: Lists the type names for all the outputs of the currently scoped SuperBlock. Obtained from the Output User Type field of the derived block **Outputs** tab. All values for a SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_unit_ls

Description: Lists the units of measure of the output channels of the currently scoped SuperBlock. These values come from the Output Unit field of the derived block **Document** tab. All output units for the SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_extout_wsize_li

Description: Lists the word sizes for all the outputs of the currently scoped SuperBlock. Obtained from the Word Size field of the SuperBlock **Attributes** tab. All values for a SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_freq_r

Description: The frequency rate of execution for a discrete SuperBlock. The frequency is computed as the reciprocal of the Sample Interval from the currently scoped SuperBlock **Attributes** tab. The frequency rate is expressed in Hertz.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

sb_has_in_b

Description: Determines if the currently scoped SuperBlock has at least one external input.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

sb_has_in_data_b

Description: Determines if a user filled out any of the fields of the SystemBuild External Input Description Form for the currently scoped SuperBlock.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

sb_id_i

Description: ID of this SuperBlock scope.
Size: scalar
Scope Class: SuperBlock
Category: DocumentIt

sb_in_minv_ls

Description: Lists the minimum values for all the inputs of the currently scoped SuperBlock. Obtained from the Input Min field of the SuperBlock **Document** tab. All values for a SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_in_i` parameter.
Size: `num_sb_in_i`
Scope Class: SuperBlock
Category: DocumentIt

sb_is_dscr_b

Description: Determines if the currently scoped SuperBlock is of type Discrete.
Size: scalar
Scope Class: SuperBlock
Category: DocumentIt

sb_is_trig_b

Description: Determines if the currently scoped SuperBlock is of type Trigger.
Size: scalar
Scope Class: SuperBlock
Category: DocumentIt

sb_mangled_name_s

Description: Mangled name of a SuperBlock.
Size: scalar
Scope Class: SuperBlock
Category: DocumentIt

sb_mapped_vars_ls

Description: Lists the mapped vars of the currently scoped component.
Size: `num_sb_mapped_vars_i`
Scope Class: SuperBlock
Category: DocumentIt

sb_mapping_vars_ls

Description: Lists the mapping vars of the currently scoped component.
Size: `num_sb_mapping_vars_i`
Scope Class: SuperBlock
Category: DocumentIt

sb_name_s

Description: Lists the currently scoped SuperBlock. This is obtained from the Name field of the SuperBlock **Attributes** tab.
Size: scalar
Scope Class: SuperBlock
Category: DocumentIt

sb_names_ls

Description: Lists the SuperBlocks in the model.

Size: `nsupblks_i`

Scope Class: System

Category: DocumentIt

sb_out_minv_ls

Description: Lists the minimum values for all the outputs of the currently scoped SuperBlock. Obtained from the Output Minimum field of the derived block **Document** tab. All values for a SuperBlock can be retrieved by indexing this parameter within a loop on the `num_sb_out_i` parameter.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_out_post_s

Description: For a triggered SuperBlock, this is the output posting option. The value can be found in the Output Posting field of the currently scoped SuperBlock **Attributes** tab.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

sb_out_srcchn_li

Description: Lists the block channel number to which the currently scoped SuperBlock outputs are associated. This parameter can be indexed within a loop on the `num_sb_out_i` parameter to obtain all the block channel numbers associated with this SuperBlock outputs.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_out_srcid_li

Description: Lists the block IDs from which the currently scoped SuperBlock outputs are generated. This parameter can be used within a loop on the `num_sb_out_i` parameter to obtain the IDs of all the blocks which generate SuperBlock outputs.

Size: `num_sb_out_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_parent_id_i

Description: ID of the parent SuperBlock of this SuperBlock.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

sb_proc_intr_name_s

Description: Interrupt SuperBlock interrupt name.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

sb_sample_r

Description: The contents of the Sample Period field for a discrete SuperBlock or the Timing Requirement field for a triggered SuperBlock from the currently scoped SuperBlock **Attributes** tab. The sampling interval is expressed in seconds per cycle.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

sb_skew_r

Description: The contents of the first Sample Skew field of the currently scoped SuperBlock **Attributes** tab for a discrete SuperBlock. For SuperBlocks, this field represents the time skew, in seconds, for reading the external inputs.

Size: scalar

Scope Class: SuperBlock

Category: DocumentIt

sb_usr_par_ext_ls

Description: Lists the extensions associated with each user parameter in the SuperBlock.

Size: `num_sb_user_par_i`

Scope Class: SuperBlock

Category: DocumentIt

sb_usr_par_ls

Description: Lists the user parameter names as defined in the Comment section of the currently scoped SuperBlock. This parameter can be indexed within a loop on the `num_sb_user_par_i` parameter to obtain all the user parameter names within the document of the currently scoped SuperBlock.

Size: `num_sb_user_par_i`

Scope Class: SuperBlock

Category: DocumentIt

schedulingcount_li

Description: Scheduling counts of tasks.

Size: scalar

Scope Class: System

Category: AutoCode

schedulerfreq_r

Description: Frequency of the scheduler.
Size: scalar
Scope Class: System
Category: AutoCode

scheduler_priority_i

Description: Priority of the scheduler.
Size: scalar
Scope Class: System
Category: AutoCode

setenables( )

Description: Code to check for enable signals and latch.
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

settriggers( )

Description: Code to check for trigger signals and latch.
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

sgtype_b

Description: Indicates if the model has system global-scope variables.
Size: scalar
Scope Class: System
Category: AutoCode

sgtype_blk_channel_list_li

Description: Lists the starting channel (pin) of the each of the blocks in the set of blocks that write into the *n*th system global-scope variable.

Size: Contains the block numbers on which to scope in order to reach the blocks that produced a particular *true global* signal. If the name of a particular *true global* signal is found as the *k*th element of the token `@sgtype_members_ls@`, then the blocks producing this signal can be found by successively scoping to the block and SuperBlock numbers in the `@sgtype_blk_list_idx_li[k]@th` … `@sgtype_blk_list_idx_li[k+1]@th` positions of `@sgtype_blk_ordinal_li@` and `@sgtype_sb_ordinal_li@`.

Scope Class: System

Category: AutoCode

sgtype_blk_list_li

Description: Lists the block number of the set of blocks that write into the *n*th system global-scope variable. The block number is a unique number for all blocks in the entire system. Channel numbers are 0-based.

Size: Contains the block numbers on which to scope in order to reach the blocks that produced a particular *true global* signal. If the name of a particular *true global* signal is found as the *k*th element of the token `@sgtype_members_ls@`, then the blocks producing this signal can be found by successively scoping to the block and SuperBlock numbers in the `@sgtype_blk_list_idx_li[k]@th` … `@sgtype_blk_list_idx_li[k+1]@th` positions of `@sgtype_blk_ordinal_li@` and `@sgtype_sb_ordinal_li@`.

Scope Class: System

Category: AutoCode

sgtype_blk_list_idx_li

Description: Lists the starting index into the `sgtype_blk_list_li` and `sgtype_blk_channel_list_li` lists for the *n*th system global-scope variable. The last element in this list represents the total size of each of those two other lists. The following example shows how to access block/channel writer information.

```
@LOOPP n=0, n lt sgtype_nmembers_i, n = n plus 1@@
@* *@System Global-Scope Variable:
@sgtype_members_ls[n]@
@   LOOPP i=sgtype_blk_list_idx_li[n],
          i lt sgtype_blk_list_idx_li[n plus 1],
          i = i plus 1@@
@*   *@  Blk #@sgtype_blk_list_li[i]@@
@*   *@  Chn #@sgtype_blk_channel_li[i]@
@   ENDLOOPP@@
@ENDLOOPP@
```

Size: `sgtype_nmembers_i` + 1

Scope Class: System

Category: AutoCode

sgtype_blk_ordinal_li

Description: Lists the block numbers on which to scope in order to reach the blocks that produced a particular *true global* signal.

If the name of a particular *true global* signal is found as the *k*th element of the token `@sgtype_members_ls@`, then the blocks producing this signal can be found by successively scoping to the block and SuperBlock numbers in the `@sgtype_blk_list_idx_li[k]@th` … `@sgtype_blk_list_idx_li[k+1]@th` positions of `@sgtype_blk_ordinal_li@` and `@sgtype_sb_ordinal_li@`.

Note A *true global* is a uniquely named signal that is either a global/file scoped in C or declared as part of the system package in Ada.

Size: Equal to the total number of times system global variables appear as block outputs in the model. Every appearance is counted, even if the same system global is written to more than once as an output of a single block. You must use the existing token `@sgtype_blk_list_idx_li@`

to determine where the entries for a given system global begin and end within `@sgtype_blk_ordinal_li@` and `@sgtype_sb_ordinal_li@`.

Scope Class: System

Category: AutoCode

sgtype_members_init_lb

Description: Initializes the names of the system global-scope variables.

Size: `sgtype_nmembers_i`

Scope Class: System

Category: AutoCode

sgtype_members_ls

Description: Lists the names of the system global-scope variables.

Size: `sgtype_nmembers_i`

Scope Class: System

Category: AutoCode

sgtype_members_memaddr_ls

Description: Lists the memory addresses for the system global-scope variables.

Size: `gtype_nmembers_i`

Scope Class: System

Category: AutoCode

sgtype_members_size_li

Description: Lists the size of the dimension of the system global-scope variable. A value of 2 or greater indicates the variable is an array.

Size: `sgtype_nmembers_i`

Scope Class: System

Category: AutoCode

sgtype_members_typ_pfix_ls

Description: Lists the data type name of the system global-scope variables.

Size: `sgtype_nmembers_i`

Scope Class: System

Category: AutoCode

sgtype_members_typ_sfix_ls

Description: Lists the suffix of the system global scope variables.

Size: `sgtype_nmembers_i`

Scope Class: System

Category: AutoCode

sgtype_members_typ_li

Description: Lists the enumerated value of the data type of the system global-scope variables. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.

Size: `sgtype_nmembers_i`

Scope Class: System

Category: AutoCode

sgtype_members_value_ls

Description: Lists the values of the system global-scope variables.

Size: `sgtype_nmembers_i`

Scope Class: System

Category: AutoCode

sgtype_nmembers_i

Description: The total number of system global-scope variables.

Size: scalar

Scope Class: System

Category: AutoCode

sgtype_sb_ordinal_li

Description: Lists the SuperBlock numbers on which to scope in order to reach the blocks that produced a particular *true global* signal.

If the name of a particular *true global* signal is found as the *k*th element of the token `@sgtype_members_ls@`, then the blocks producing this signal can be found by successively scoping to the block and SuperBlock numbers in the `@sgtype_blk_list_idx_li[k]@th` … `@sgtype_blk_list_idx_li[k+1]@th` positions of `@sgtype_blk_ordinal_li@` and `@sgtype_sb_ordinal_li@`.

Note A *true global* is a uniquely named signal which is either a global/file scoped in C or declared as part of the system package in Ada.

Size: Equal to the total number of times system global variables appear as block outputs in the model. Every appearance is counted, even if the same system global is written to more than once as an output of a single block. You must use the existing token `@sgtype_blk_list_idx_li@` to determine where the entries for a given system global begin and end within `@sgtype_blk_ordinal_li@` and `@sgtype_sb_ordinal_li@`.

Scope Class: System

Category: AutoCode

small_frame( )

Description: A TPL function that can generate an anchored frame of size 5 × 5 in. for FrameMaker target documents. An empty box appears when you invoke the document. You can import `.eps` images into the anchored frame using the Capture feature in the application. Refer to the *large_frame( )* section.

Size: scalar

Scope Class: System

Category: AutoCode

smco_option_b

Description: Indicates if the `-smco` option is specified.

Size: scalar

Scope Class: System

Category: AutoCode

sp_fxp_argtype_li

Description: Encoded data of the types of the left and right arguments to the overloaded operators. A list containing `FXPPACKEDID` values representing the data types of the left and right arguments for the operator.

Unary operators only use the right type, except the ABS operator uses the left type. Refer to the *Special Type Encodings* section.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

sp_fxp_conversionid_li

Description: Lists the type of conversions to generate. A list containing values that represent the type of conversion. The value is used as an index for `fxp_conversions_ls` parameter. The value has the range of –1, 0..17, 1000..1017. A value greater than or equal to 1000 represents a duplicated conversion function. This means that at some other index in this list, the exact same conversion exists. To obtain which conversion is duplicated, subtract 1000. A value of –1 is not a fixed-point conversion and must not be generated.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

sp_fxp_convertarg_li

Description: Encoded data of the types used for the conversion. A list of `FXPPACKEDID` values that represent the fixed-point type(s) used in the conversion. This type is in the right type. Most conversions use only one fixed-point type, either as the type of the argument or the type of the result. Some conversions require two fixed-point types. In those cases, the right type is the argument type and the left type is the result type. Refer to the *Special Type Encodings* section.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

sp_fxp_operatorid_li

Description: Lists the operators to overload. A list of indices for use with the `fxp_operator_list_ls` and `fxp_operator_name_ls` parameters. Values of the index range from –1, 0..10, 1000..1010. An index greater than or equal to 1000 represents that the operator is a duplicate; that is, at some other index in the list, the same operator with left, right and result type information exists. To find the duplicate operator, subtract 1000 from the number. An index with the value of –1 is not a fixed-point operator and must be skipped.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

sp_fxp_resulttype_li

Description: Encoded data of the type of the result of the overload operators. A list containing `FXPPACKEDID` values representing the result type of the operator. The result data type is in the left packed ID. The right packed ID is reserved for an intermediate fixed-point type. Currently, only addition and subtraction operators use the intermediate type. Refer to the *Special Type Encodings* section.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

sp_nfxp_conversions_i

Description: Number of conversion functions to generate. The *n*th element of this list parameter and `nfxp_conversions_i` represent data pertaining to the *n*th conversion function that is to be generated.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

sp_nfxp_opfunctions_i

Description: Number of overloaded operator functions to generate. The *n*th element of this list parameter and `nfxp_opfunctions_i` represent data pertaining to the *n*th overloaded function that is to be generated.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

srate_opti_b

Description: True if single rate optimization has occurred which makes the system external input as the task input and the task output as system output. This is done when `doublebuf_b` is false and when there are no DataStores or disconnected signals in the model.

Size: scalar

Scope Class: System

Category: AutoCode

ssfreq_lr

Description: Frequencies of the tasks.

Size: `ntasks_i`

Scope Class: System

Category: AutoCode

ssid_li

Description: Lists the task IDs of all the tables assigned to this processor.

Size: `pe_ntabks_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

ssmode_flags_ls

Description: Lists the mode flag values for each subsystem. The *sspriority_li* section shows the usage of this list. For RTOS configuration information, refer to the *AutoCode Reference*.

Size: `ntasks_i`

Scope Class: System

Category: AutoCode

ssoptime_lr

Description: Output time of the tasks.

Size: `ntasks_i`

Scope Class: System

Category: AutoCode

sspriority_i

Description: Priority of the subsystem.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

sspriority_li

Description: Lists the priority values for each subsystem. For RTOS configuration information, refer to Chapter 4, *Generating Code for Real-Time Operating Systems*, of the *AutoCode Reference*. The following example shows the usage of this token with TPL code accessing RTOS configuration data:

```
@INT i,j@
Scheduler Priority: @scheduler_priority_i@

@LOOPP i=0, i lt nprocessors_i, i=i plus 1@@
@j=i plus 1@@
Processor #@j@   IPNumber: @prsr_ip_name_ls[i]@
@ENDLOOPP@@
```

```
@LOOPP i=0, i lt ntasks_i, i=i plus 1@@
@  j=i plus 1@@
Task @j@
    PRIORITY: @sspriority_li[i]@
    STACK SZ: @ssstack_size_li[i]@
    PRSR    : @ssprocessor_map_li[i]@
    MODE FLG: @ssmode_flags_ls[i]@

@ENDLOOPP@@

standard procs: @nstandard_procs_i@
startup       : @nstartup_procs_i@
backgnd       : @nbackgnd_procs_i@
interrupt     : @ninterrupt_procs_i@

@LOOPP i=0, i lt nstartup_procs_i, i=i plus 1@@
@j = startup_procs_li[i]@@
@SCOPE PROCEDURE j@@
Startup Procedure #@i@: @procedurename_s@
@j = j minus nstandard_procs_i@@
    PRIORITY: @proc_priority_li[j]@
    STACK SZ: @proc_stack_size_li[j]@
    PRSR    : @proc_processor_map_li[j]@
    MODE FLG: @proc_mode_flags_ls[j]@
    ORDER   : @proc_ordering_li[j]@

@ENDLOOPP@@
@LOOPP i=0, i lt nbackgnd_procs_i, i=i plus 1@@
@j = backgnd_procs_li[i]@@
@SCOPE PROCEDURE j@
Backgnd Procedure #@i@: @procedurename_s@
@j = j minus nstandard_procs_i@@
    PRIORITY: @proc_priority_li[j]@
    STACK SZ: @proc_stack_size_li[j]@
    PRSR    : @proc_processor_map_li[j]@
    MODE FLG: @proc_mode_flags_ls[j]@
    ORDER   : @proc_ordering_li[j]@
@ENDLOOPP@@
@LOOPP i=0, i lt ninterrupt_procs_i, i=i plus 1@@
@j = interrupt_procs_li[i]@@
@SCOPE PROCEDURE j@@
Interrupt Procedure #@i@: @procedurename_s@
@j = j minus nstandard_procs_i@@
    PRIORITY: @proc_priority_li[j]@
```

```
        STACK SZ: @proc_stack_size_li[j]@
        PRSR    : @proc_processor_map_li[j]@
        MODE FLG: @proc_mode_flags_ls[j]@
        ORDER   : @proc_ordering_li[j]@
        VECTOR  : @proc_vector_li[j]@

@ENDLOOPP@@
```

Size: ntasks_i

Scope Class: System

Category: AutoCode

ssprocessor_map_li

Description: Lists the processor assignment values for each subsystem. Refer to the *sspriority_li* section for usage of this list. For RTOS information, refer to Chapter 4, *Generating Code for Real-Time Operating Systems*, of the *AutoCode Reference*.

Size: ntasks_i

Scope Class: System

Category: AutoCode

ssskew_lr

Description: Skews of the tasks.

Size: ntasks_i

Scope Class: System

Category: AutoCode

ssstack_size_li

Description: Lists the stack size values for each subsystem. For RTOS information, refer to Chapter 4, *Generating Code for Real-Time Operating Systems*, of the *AutoCode Reference*.

Size: ntasks_i

Scope Class: System

Category: AutoCode

sstypes_ls

Description: Types of tasks.
Size: `ntasks_i`
Scope Class: System
Category: AutoCode

standard_procs_li

Description: Lists the global number for each standard procedure SuperBlock. Use the global number for the argument to SCOPE PROCEDURE.
Size: `nstandard_procs_i`
Scope Class: System
Category: AutoCode

startcount_li

Description: Starts the count of tasks. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.
Size: `ntasks_i`
Scope Class: System
Category: AutoCode

startup_procs_li

Description: Lists the global number for each startup procedure SuperBlock. Use the global number for the argument to SCOPE PROCEDURE.
Size: `nstartup_procs_i`
Scope Class: System
Category: AutoCode

std_init_bbl_i

Description: The initial bubble of the currently scoped STD block.

Size: scalar

Scope Class: STD Block

Category: DocumentIt

STRCMP(string1, string2)

Description: Returns 1 if strings are equal, otherwise returns 0. The `string1` and `string2` are string expressions which are either simple string constants or expressions formed with cat operator(s), string constants, string parameters, and dereferenced stringlist parameters. For an example of how to use this function, refer to the *String Manipulation Functions* section of Chapter 2, *Template Programming Language*.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

STRLEN(string1)

Description: Returns the length of the string. The `string1` is a string expression which is either a simple string constant or expression formed with cat operator(s), string constants, string parameters and dereferenced stringlist parameters. For an example of how to use this function, refer to the *String Manipulation Functions* section of Chapter 2, *Template Programming Language*.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

STRLOWER(string1)

Description: Returns the contents of string1 with alphabetic characters converted to lowercase. The `string1` is a string expression which is either a simple string constant or expression formed with cat operator(s), string constants, string parameters, and dereferenced stringlist parameters. For an example of how to use this function, refer to the *String Manipulation Functions* section of Chapter 2, *Template Programming Language*.

Size: string

Scope Class: TPL Library Function

Category: AutoCode

STRNCMP(string1, string2)

Description: Returns 1 if the strings compare by at most the number of characters in `string2`. If not, return 0. The `string1` and `string2` are string expressions which are either simple string constants or expressions formed with cat operator(s), string constants, string parameters, and dereferenced stringlist parameters. For an example of how to use this function, refer to the *String Manipulation Functions* section of Chapter 2, *Template Programming Language*.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

STRSTR(string1, string2)

Description: When used in an assignment statement, returns the first occurrence of the `string2` within `string1`. If not found, return an empty string. The `string1` and `string2` are string expressions which are either simple string constants or expressions formed with cat operator(s), string constants, string parameters, and dereferenced stringlist parameters. When used in an IFF statement, returns 0 or 1 if `string2` is part of `string1`. For an example of how to use this function, refer to the *String Manipulation Functions* section of Chapter 2, *Template Programming Language*.

Size: string

Scope Class: TPL Library Function

Category: AutoCode

STRUPPER(string1)

Description: Returns the contents of `string1` with all alphabetic characters converted to uppercase. The `string1` is a string expression which is either a simple string constant or expression formed with cat operator(s), string constants, string parameters, and dereferenced stringlist parameters. For an example of how to use this function, refer to the *String Manipulation Functions* section of Chapter 2, *Template Programming Language*.

Size: string

Scope Class: TPL Library Function

Category: AutoCode

struct_map( )

Description: Prints the structures/records map.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

stype_b

Description: True if this subsystem/procedure has the states structure/record declaration.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

stype_members_ls

Description: Lists the member names of the statetype structure/record.

Size: `stype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

stype_members_size_li

Description: Lists the sizes of the dimension of the member variables of the statetype structure/record. A value of 2 or greater indicates an array.

Size: `stype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

stype_members_typ_li

Description: Lists the enumerated value of the data type of the member variables of the statetype structure/record. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.

Size: `stype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

stype_nmembers_i

Description: Number of members in the structure/record representing the statetype.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

stype_tag_s

Description: The name of the statetype struct tag or record.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

subsysid_i

Description: Subsystem/Procedure ID.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

subsys_sampletime_r

Description: Sample time of the subsystem.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

sutype_b

Description: True if the system has the external input structure/record declaration.
Size: scalar
Scope Class: System
Category: AutoCode

sutype_members_ls

Description: Lists the member names of the system external input structure/record.
Size: `sutype_nmembers_i`
Scope Class: System
Category: AutoCode

sutype_members_size_li

Description: Lists the size of the system inputs. A value of 2 or greater indicates the variable is an array.

Size: `sutype_nmembers_i`

Scope Class: System

Category: AutoCode

sutype_members_typ_li

Description: Lists the enumerated value of the system input data type. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.

Size: `sutype_nmembers_i`

Scope Class: System

Category: AutoCode

sutype_members_typ_pfix_ls

Description: Lists the data type names of the system inputs.

Size: `sutype_nmembers_i`

Scope Class: System

Category: AutoCode

sutype_members_typ_sfix_ls

Description: Lists the system input names and corresponding suffixes.

Size: `sutype_nmembers_i`

Scope Class: System

Category: AutoCode

sutype_nmembers_i

Description: Number of members within the system external input structure/record.

Size: scalar

Scope Class: System

Category: AutoCode

sutype_obj_s

Description: The name of the object representing the system external input definitions.

Size: scalar

Scope Class: System

Category: AutoCode

sutype_tag_s

Description: The name of the system external input type `struct` tag or record.

Size: scalar

Scope Class: System

Category: AutoCode

sys_extin_copy( )

Description: Prints code to copy-in from ExtIn array.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

sys_extout_copy( )

Description: Prints code to copy-out to ExtOut array.
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

system_data_init( )

Description: Generates code for initializing the application data depending on the current processor scope.
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

system_output( )

Description: Prints code to post system outputs.
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

sytype_b

Description: True if the system has the external output structure/record declaration.
Size: scalar
Scope Class: System
Category: AutoCode

sytype_members_ls

Description: Lists the member names of the system external output structure/record.

Size: `sytype_nmembers_i`

Scope Class: System

Category: AutoCode

sytype_members_rid_li

Description: Lists the task IDs for each member of the system external output structure/record. The ID represents the task (subsystem) that the output comes from. A value of less than 0 indicates that the member is disconnected within the model.

Size: `sytype_nmembers_i`

Scope Class: System

Category: AutoCode

sytype_members_size_li

Description: Lists the size of the system outputs. A value of two or greater indicates the variable is an array.

Size: `sytype_nmembers_i`

Scope Class: System

Category: AutoCode

sytype_members_typ_li

Description: Lists the enumerated value of the system output data types. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.

Size: `sytype_nmembers_i`

Scope Class: System

Category: AutoCode

sytype_members_typ_pfix_ls

Description: Lists the data type names of the system outputs.
Size: `sytype_nmembers_i`
Scope Class: System
Category: AutoCode

sytype_members_typ_sfix_ls

Description: Lists the system output names and the corresponding suffixes.
Size: `sytype_nmembers_i`
Scope Class: System
Category: AutoCode

sytype_nmembers_i

Description: Number of members within the system external output structure/record.
Size: scalar
Scope Class: System
Category: AutoCode

sytype_obj_s

Description: The name of the object representing the system external output definition.
Size: scalar
Scope Class: System
Category: AutoCode

sytype_tag_s

Description: The name of the system external output type `struct` tag or record.
Size: scalar
Scope Class: System
Category: AutoCode

timeref_b

Description: Indicates if a subsystem uses `TIME` variable or not.
Size: scalar
Scope Class: Subsystem
Category: AutoCode

timerequired_b

Description: Indicates if the system uses `TIME` variable or not.
Size: scalar
Scope Class: System
Category: AutoCode

tmpversion_i

Description: Version number of the template.
Size: scalar
Scope Class: System
Category: AutoCode

to_bbl_li

Description: Lists the IDs of the *to* bubbles for each of the transitions in the currently scoped STD block. All values for the STD block can be retrieved by indexing this parameter within a loop on `num_bbls_in_std_i` while offsetting the index by using `bbl_trn_off_li`.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

toggle_rwbuffers( )

Description: Code to toggle task read/write buffers. In multiprocessor, it toggles the read and write pointers in the processor with ID zero. It computes the current read and write pointers in other processors.

Size: scalar

Scope Class: TPL Library Function

Category: AutoCode

tpl_fname_s

Description: The name of the TPL file compiled into current `.dac` file.

Size: scalar

Scope Class: System

Category: AutoCode

tpl_createdate_s

Description: The creation date of the TPL file compiled into the current `.dac` file.

Size: scalar

Scope Class: System

Category: AutoCode

trig_source_i

Description: Number indicates the source of the triggered or enabled signal.

Size: scalar

Scope Class: Subsystem

Category: AutoCode

trn_cmt_ext_ls

Description: Lists the extensions associated with all transition comment fields.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

trn_cmt_ls

Description: Lists the textual descriptions of the transitions for each of the bubbles in the currently scoped STD block. All values for the STD block can be retrieved by indexing this parameter within a loop on `num_bbls_in_std_i` while offsetting the index by using `bbl_trn_off_li`.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

trn_has_out_lb

Description: Lists whether the transitions have outputs for each of the bubbles in the currently scoped STD block. All values for the STD block can be retrieved by indexing this parameter within a loop on `num_bbls_in_std_i` while offsetting the index by using `bbl_trn_off_li`.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

trn_name_ls

Description: Lists the names of the transitions for each of the bubbles in the currently scoped STD block. All values for the STD block can be retrieved by indexing this parameter within a loop on `num_bbls_in_std_i` while offsetting the index by using `bbl_trn_off_li`.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

trn_priority_li

Description: Lists the priorities of the transitions for each of the bubbles in the currently scoped STD block. All values for the STD block can be retrieved by indexing this parameter within a loop on `num_bbls_in_std_i` while offsetting the index by using `bbl_trn_off_li`.

Size: `num_bbls_in_std_i`

Scope Class: STD Block

Category: DocumentIt

ucb_names_ls

Description: Lists the names of the UCBs used within the scoped subsystem or procedure.

Size: `nucbs_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

undefined_type_i

Description: Undefined data type for a %var or variable block variable.

Size: scalar

Scope Class: System

Category: AutoCode

uni_sb_li

Description: Lists the unique SuperBlock IDs.
Size: `num_uni_sb_i`
Scope Class: System
Category: DocumentIt

update_dsext( )

Description: Code to update DataStores from external inputs.
Size: scalar
Scope Class: TPL Library Function
Category: AutoCode

userblock_b

Description: True if the model contains any UserCode Block.
Size: scalar
Scope Class: System
Category: AutoCode

user_param ("string1", "string2")

Description: Used to create user-definable template parameters. `string1` is the name of any user parameter declared in the currently scoped SuperBlock, DataStore, or STD and `string2` can only be string BLOCK, SUPERBLOCK, or DATASTORE denoting the scope classes.
Size: scalar
Scope Class: TPL Library Function
Category: DocumentIt

user_type_i

Description: A user-defined data type for a %var or variable block variable.

Size: scalar

Scope Class: System

Category: AutoCode

usertype_basename_ls

Description: An array of strings that contain the names of all the base types of the user types of the model.

Size: `n_user_defined_type_i`

Scope Class: System

Category: AutoCode

usertype_name_ls

Description: An array of strings that contain the names of the user types in the model.

Size: `n_user_defined_type_i`

Scope Class: System

Category: AutoCode

utype_b

Description: True if the subsystem/procedure has the input structure/record declaration.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

utype_members_ls

Description: Lists member names of the subsystem/procedure input structures/record.

Size: `utype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

utype_members_size_li

Description: Lists the size of the subsystem inputs. A value of 2 or greater indicates the variable is an array.

Size: `utype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

utype_members_typ_li

Description: Lists the enumerated value of the subsystem input data types. Compare the values against the set of data type tokens (`*_type_i`) to determine the data type.

Size: `utype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

utype_members_typ_pfix_ls

Description: Lists the data type names of the subsystem outputs.

Size: `utype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

utype_members_typ_sfix_ls

Description: Lists the suffix of the subsystem input names.
Size: `utype_nmembers_i`
Scope Class: Subsystem/Procedure
Category: AutoCode

utype_nmembers_i

Description: Number of members within the subsystem/procedure input structure/record.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

utype_obj_s

Description: The name of the object representing the subsystem input definitions.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

utype_tag_s

Description: The name of the subsystem/procedure input type struct tag or record.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

vars_ls

Description: Lists the names of all the variable block variables in the model. The following example shows the usage of this token where TPL code displays the data type of variable block variables:

```
@INT i@

@LOOPP i=0, i lt nvars_i, i=i plus 1@@
    @vars_ls[i]@ is @@
@IFF vars_type_li[i] eq float_type_i@@
a float type!
@ENDIFF@@
@IFF vars_type_li[i] eq integer_type_i@@
an integer type!
@ENDIFF@@
@IFF vars_type_li[i] ge fixed_unsigned_byte_type_i and
     vars_type_li[i] le fixed_signed_long_type_i@@
a fixed-point type with radix @vars_subtype_li[i]@
@ENDIFF@@
@ENDLOOPP@@
```

Size: `nvars_i`

Scope Class: System

Category: AutoCode

vars_prsr_scope_li

Description: Lists the values indicating the processor scope of each of the variable block variables in the model relative to the last processor scoped upon. A value of 0 indicates not local to current processor and not shared; 1 indicates local to current processor and not shared; 2 indicates not local to processor but shared; and 3 indicates local to processor and shared.

Size: `nvars_i`

Scope Class: System

Category: AutoCode

vars_sfix_dim_start_li

Description: An index into the start of the `vars_typ_sfix_li` for each of the variable block variables.

Size: `nvars_i`

Scope Class: System

Category: AutoCode

vars_subsys_freq_li

Description: Lists the values representing the frequency of a variable block for each subsystem and procedure it is used in. Use `vars_subsys_idx_li` to index to the start of the *n*th variable block.

Size: Sum of all subsystems and procedures which use a variable block for each variable block in the model.

Scope Class: System

Category: AutoCode

vars_subsys_idx_li

Description: Lists the starting index for the *n*th variable block variable into the `vars_subsys_freq_li` and `vars_subsys_li` lists. The `nvars+1th` element represents the total size of the `subsys` and `freq` lists. The following example shows the usage of this token where TPL code gets usage information about variable block variables:

```
@LOOPP i=0, i lt nvars_i, i = i plus 1@@
@* *@variable block: @vars_ls[i]@
@    LOOPP j=vars_subsys_idx_li[i],
           j lt vars_subsys_idx_li[i plus 1],
           j = j plus 1@@
@*   *@ subsys/proc ID: @vars_subsys_li[j]@@
@*   *@ freq: @vars_subsys_freq_li[j]@
@    ENDLOOPP@
@ENDLOOPP@
```

Size: `nvars_i+1`

Scope Class: System

Category: AutoCode

vars_subsysli

Description: Lists the numbers representing which subsystem(s) and procedure(s) a variable block is used in. Each usage is placed consecutively within the list. Use `vars_subsys_idx_li` to find the start within this list for the *n*th variable block. Numbers are assigned to subsystems and procedures in the following order: subsystems, standard procedures, startup procedures, background procedures, and interrupt procedures.
In addition, startup procedure numbers are represented as negative values. The values are the negation of their ordering number.

Size: Sum of all subsystems and procedures which use a variable block for each variable block in the model.

Scope Class: System

Category: AutoCode

vars_subtype_li

Description: Subtype information for the variable block variables. If the type is non-fixed point, the value is 0. If fixed-point, the value is the radix of the type.

Size: `nvars_i`

Scope Class: System

Category: AutoCode

vars_typ_pfix_ls

Description: Lists the type specification of each of the variable block variables used in the model.

Size: `nvars_i`

Scope Class: System

Category: AutoCode

vars_typ_sfix_dim_li

Description: Lists the number of dimensions for each variable block variable used in the model. Scalar variables have a dimension of zero (0).

Size: `nvars_i`

Scope Class: System

Category: AutoCode

vars_typ_sfix_li

Description: The size of each dimension for each variable block variable used in the model. If variable is scalar, its dimension is considered 0.

Size: Sum of all dimensions for all variable blocks. A scalar var counts as one in the list count.

Scope Class: System

Category: AutoCode

vars_type_li

Description: An enumerated value representing the type of the variable block variable.

The elements in the list can be compared to the following parameters, which each represent a specific data type:

`logical_type_i, fixed_signed_long_i, fixed_unsigned_long_i, fixed_signed_short_i, fixed_unsigned_short_i, fixed_signed_byte_i, fixed_unsigned_byte_i, and user_type_i.`

Refer to the *vars_ls* section for the usage of this list.

Size: `nvars_i`

Scope Class: System

Category: AutoCode

vbco_option_b

Description: Indicates if the `-vbco` option is specified.

Size: scalar

Scope Class: System

Category: AutoCode

ytype_b

Description: True if the subsystem/procedure has the output structure/record declaration.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

ytype_members_ls

Description: Lists the member names of the subsystem/procedure output structure/record.

Size: `ytype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

ytype_members_size_li

Description: Lists the size of the subsystem outputs. A value of two or greater indicates the variable is an array.

Size: `ytype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

ytype_members_typ_li

Description: Lists the enumerated value of the subsystem output data types. Compare the values against the set of data type tokens (`*_type_i`) to determine the type.

Size: `ytype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

ytype_members_typ_pfix_ls

Description: Lists the data type names of the subsystem outputs.

Size: `ytype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

ytype_members_typ_sfix_ls

Description: Lists the suffix of the subsystem output names.

Size: `ytype_nmembers_i`

Scope Class: Subsystem/Procedure

Category: AutoCode

ytype_nmembers_i

Description: Number of members within the subsystem/procedure output structure/record.

Size: scalar

Scope Class: Subsystem/Procedure

Category: AutoCode

ytype_obj_s

Description: The name of the object representing the subsystem output definition.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

ytype_tag_s

Description: The name of the subsystem/procedure output `struct` tag or record.
Size: scalar
Scope Class: Subsystem/Procedure
Category: AutoCode

A

AutoCode TPL Tokens Listed by SCOPE Class

This appendix lists all of the TPL tokens for AutoCode by SCOPE class. Details are not provided in this appendix. Rather, each parameter is cross-referenced to Chapter 3, *TPL Token Reference*, where a full description of each token is given.

The AutoCode SCOPE classes are:

- System
- DataStore
- Processor
- Subsystem
- Procedure
- Subsystem/Procedure
- Library Function

Table A-1. AutoCode System Scope Class Parameters

Parameter	Parameter
backgnd_procs_li	busy_ds_registers_i
childprocs_ins_off_li	childprocs_ins_li
childprocs_off_li	childprocs_li
continuous_b	continuous_id_i
currentdate_s	currentfname_pfix_s
currentfname_s	dac_createdate_s
dac_fname_s	desprocs_off_li
desprocs_li	doublebuf_b
epsilon_r	errorcheck_option_b
extended_procedure_info_b	fast_scheduler_b

Table A-1. AutoCode System Scope Class Parameters (Continued)

Parameter	Parameter
fixed_signed_byte_type_i	fixed_signed_long_type_i
fixed_signed_short_type_i	fixed_unsigned_byte_type_i
fixed_unsigned_long_type_i	fixed_unsigned_short_type_i
fixpt_math_b	float_type_i
full_uy_required_b	fxp_argtype_li
fxp_conversionid_li	fxp_conversions_ls
fxp_convertarg_li	fxp_long_fxpname_ls
fxp_operatorid_li	fxp_operator_list_ls
fxp_operator_name_ls	fxp_resulttype_li
fxp_short_fxpname_ls	initialtaskstate_ls
integer_type_i	integrator_i
interrupt_procs_li	kroption_b
language_s	logical_type_i
macro_procs_ls	modelname_s
multiprocessor_b	n_user_defined_type_i
nbackgnd_procs_i	nchildprocs_ins_li
nchildprocs_li	ndesprocs_li
nds_i	nenabled_i
nfxp_conversions_i	nfxp_opfunctions_i
ninterrupt_procs_i	nmacro_procs_i
nobusmap_b	noerr_option_b
noicmap_b	norestart_b
npercentvars_i	nperiodic_i
nprocedures_i	nprocessors_i
nstandard_procs_i	nstartup_procs_i
ntasks_i	ntriggered_i

Table A-1. AutoCode System Scope Class Parameters (Continued)

Parameter	Parameter
numin_i	numout_i
nvars_i	ordered_procs_li
outputcount_li	outputfile_dir_s
outputfile_ext_s	outputfile_pfix_s
output_fname_s	percentvars_ls
percentvars_as_varblk_li	percentvars_prsr_scope_li
percentvars_sfix_dim_start_li	percentvars_subtype_li
percentvars_typ_pfix_ls	percentvars_typ_sfix_li
percentvars_typ_sfix_dim_li	percentvars_type_li
processorid_li	proc_mode_flags_ls
proc_ordering_li	proc_priority_li
proc_processor_map_li	proc_stack_size_li
proc_vector_li	procs_only_b
prsr_ip_name_ls	pseudorate_b
returnvalue_s	rtf_fname_s
rtos_option_b	schedulingcount_li
schedulerfreq_r	scheduler_priority_i
sgtype_b	sgtype_blk_channel_list_li
sgtype_blk_list_li	sgtype_blk_list_idx_li
sgtype_blk_ordinal_li	sgtype_members_init_lb
sgtype_members_ls	sgtype_members_memaddr_ls
sgtype_members_size_li	sgtype_members_typ_pfix_ls
sgtype_members_typ_sfix_ls	sgtype_members_typ_li
sgtype_members_value_ls	sgtype_nmembers_i
sgtype_sb_ordinal_li	smco_option_b
srate_opti_b	ssfreq_lr

Table A-1. AutoCode System Scope Class Parameters (Continued)

Parameter	Parameter
ssmode_flags_ls	ssoptime_lr
sspriority_li	ssprocessor_map_li
ssskew_lr	ssstack_size_li
sstypes_ls	standard_procs_li
startcount_li	startup_procs_li
sutype_b	sutype_members_ls
sutype_members_size_li	sutype_members_typ_li
sytype_members_typ_pfix_ls	sytype_members_typ_sfix_ls
sutype_nmembers_i	sutype_obj_s
sutype_tag_s	stype_b
stype_members_ls	sytype_members_rid_li
sytype_members_size_li	sytype_members_typ_li
sytype_members_typ_pfix_ls	sutype_members_typ_sfix_ls
sytype_nmembers_i	sytype_obj_s
sytype_tag_s	timerequired_b
tmpversion_i	tpl_fname_s
tpl_createdate_s	undefined_type_i
userblock_b	user_type_i
usertype_basename_ls	usertype_name_ls
vars_ls	vars_prsr_scope_li
vars_sfix_dim_start_li	vars_subsys_freq_li
vars_subsys_idx_li	vars_subsysli
vars_subtype_li	vars_typ_pfix_ls
vars_typ_sfix_dim_li	vars_typ_sfix_li
vars_type_li	vbco_option_b

Table A-2. AutoCode DataStore Scope Class Parameters

Parameter	Parameter
ds_id_i	dstype_b
dstype_members_ls	dstype_nmembers_i
dstype_obj_s	dstype_tag_s

Table A-3. AutoCode Processor Scope Class Parameters

Parameter	Parameter
pe_backgnd_procs_li	pe_id_i
pe_interrupt_procs_li	pe_macro_procs_ls
pe_nbackgnd_procs_i	pe_ninterrupt_procs_i
pe_nmacro_procs_i	pe_npercentvars_i
pe_nprocedures_i	pe_nstandard_procs_i
pe_nstartup_procs_i	pe_ntasks_i
pe_nvars_i	pe_ordered_procs_li
pe_percentvar_map_li	pe_standard_procs_li
pe_startup_procs_li	pe_var_map_li

Table A-4. AutoCode Subsystem Scope Class Parameters

Parameter	Parameter
initop_b	reset_b
timeref_b	—

Table A-5. AutoCode Procedure Scope Class Parameters

Parameter	Parameter
errorcheck_option_b	full_info_b
proc_info_parameters_b	proc_info_percentvars_b
proc_initneeded_b	proc_intr_name_s

Table A-6. AutoCode Subsystem/Procedure Scope Class Parameters

Parameter	Parameter
envflags_i	full_uy_required_b
gtype_b	gtype_blk_channel_list_li
gtype_blk_list_li	gtype_blk_ordinal_li
gtype_members_init_lb	gtype_members_ls
gtype_members_memaddr_ls	gtype_members_size_li
gtype_members_typ_li	gtype_members_typ_pfix_ls
gtype_members_typ_sfix_ls	gtype_members_value_ls
gtype_nmembers_i	gtype_sb_ordinal_li
init_call_required_b	inputs_i
itype_b	itype_members_ls
itype_members_size_li	itype_members_typ_li
itype_nmembers_i	itype_tag_s
npercentvars_used_i	nprocs_i
nucbs_i	numip_i
numrp_i	numxs_i
numiparsym_i	numrparsym_i
outputs_i	percentvars_used_li
procedurename_s	processor_i
proc_id_li	proc_names_ls

Table A-6. AutoCode Subsystem/Procedure Scope Class Parameters (Continued)

Parameter	Parameter
proc_nsbs_i	proc_sb_id_li
proc_userblock_b	pstype_b
pstype_members_ls	pstype_members_size_li
pstype_members_typ_li	pstype_nmembers_i
pstype_tag_s	sp_fxp_argtype_li
sp_fxp_conversionid_li	sp_fxp_convertarg_li
sp_fxp_operatorid_li	sp_fxp_resulttype_li
sp_nfxp_conversions_i	ssid_li
sspriority_i	stype_b
stype_members_ls	stype_members_size_li
stype_members_typ_li	stype_nmembers_i
stype_tag_s	subsysid_i
subsys_sampletime_r	ucb_names_ls
utype_b	utype_members_typ_li
utype_members_size_li	utype_members_typ_li
utype_members_typ_pfix_ls	utype_members_typ_sfix_ls
utype_nmembers_i	utype_obj_s
utype_tag_s	ytype_b
ytype_members_ls	ytype_members_size_li
ytype_members_typ_li	ytype_members_typ_pfix_ls
ytype_members_typ_sfix_ls	ytype_nmembers_i
ytype_obj_s	ytype_tag_s

Table A-7. AutoCode TPL Library Functions

Parameter	Parameter
ATOF(string)	ATOI(string)
call_cont_init( )	call_cont_subsys( )
collect_fxpdata( )	declare_cont_info( )
declare_cont_init( )	declare_cont_states( )
declare_dczeros( )	declare_globals( )
declare_percentvars( )	declare_procs( )
declare_structs( )	declare_subsystem( )
declare_varblocks( )	define_cont_info( )
define_cont_init( )	define_cont_states( )
define_dczeros( )	define_percentvars( )
define_procs( )	define_structs( )
define_subsystem( )	define_varblocks( )
dispatch_subsys( )	get_pvar_init(pvaridx {, validx})
get_varblk_init(varblkidx {, validx})	include_img( )
large_frame( )	proc_ucb_hook( )
sample_hold({subsysId})	setenables( )
settriggers( )	small_frame( )
struct_map( )	STRCMP(string1, string2)
STRLEN(string1)	STRLOWER(string1)
STRNCMP(string1, string2)	STRSTR(string1, string2)
STRUPPER(string1)	struct_map( )
sys_extin_copy( )	sys_extout_copy( )
system_data_init( )	system_output( )
toggle_rwbuffers( )	update_dsext( )

B

DocumentIt TPL Tokens Listed by SCOPE Class

This appendix lists all of the TPL tokens for DocumentIt by SCOPE class. Details are not provided in this appendix. Rather, the list is cross-referenced to Chapter 3, *TPL Token Reference*, where a full description of each token is given.

The DocumentIt SCOPE classes are:

- System
- SuperBlock
- DataStore
- Block
- STD
- Library Function

Table B-1. DocumentIt System Parameters

Parameter	Parameter
currentdate_s	currentfname_pfix_s
currentfname_s	dac_createdate_s
dac_fname_s	ds_names_ls
modelname_s	nds_i
nsupblks_i	numin_i
numout_i	num_uni_sb_i
outputfile_dir_s	outputfile_ext_s
outputfile_pfix_s	output_fname_s
rtf_fname_s	sb_names_ls
uni_sb_li	—

Table B-2. DocumentIt SuperBlock Parameters

Parameter	Parameter
level_i	num_blks_in_sb_i
num_sb_in_i	num_sb_mapped_vars_i
num_sb_mapping_vars_i	num_sb_out_i
num_sb_usr_par_i	sb_actv_sig_s
sb_attr_s	sb_cmt_ext_s
sb_cmt_s	sb_code_cmt_s
sb_extin_accu_lr	sb_extin_accu_ls
sb_extin_dsc_ls	sb_extin_id_ls
sb_extin_maxv_lr	sb_extin_maxv_ls
sb_extin_minv_lr	sb_extin_minv_ls
sb_extin_mnem_ls	sb_extin_name_ls
sb_extin_radix_li	sb_extin_scope_ls
sb_extin_typ_li	sb_extin_typ_ls
sb_extin_typ_name_ls	sb_extin_unit_ls
sb_extin_wsize_li	sb_extout_accu_lr
sb_extout_accu_ls	sb_extout_dsc_ls
sb_extout_id_ls	sb_extout_maxv_lr
sb_extout_maxv_ls	sb_extout_memaddr_ls
sb_extout_minv_lr	sb_extout_mnem_ls
sb_extout_name_ls	sb_extout_scope_ls
sb_extout_radix_li	sb_extout_typ_li
sb_extout_typ_ls	sb_extout_typ_name_ls
sb_extout_unit_ls	sb_extout_wsize_li
sb_freq_r	sb_has_in_b
sb_has_in_data_b	sb_id_i
sb_in_minv_ls	sb_is_dscr_b

Table B-2. DocumentIt SuperBlock Parameters (Continued)

Parameter	Parameter
sb_is_trig_b	sb_mapped_vars_ls
sb_mapping_vars_ls	sb_name_s
sb_out_minv_ls	sb_out_post_s
sb_out_srcchn_li	sb_out_srcid_li
sb_parent_id_i	sb_proc_intr_name_s
sb_sample_r	sb_skew_r
sb_usr_par_ext_ls	sb_usr_par_ls

Table B-3. DocumentIt DataStore Parameters

Parameter	Parameter
ds_cmt_s	ds_cmt_ext_s
ds_code_cmt_s	ds_id_i
ds_name_s	ds_reg_accu_lr
ds_reg_accu_ls	ds_reg_dsc_ls
ds_reg_initv_lr	ds_reg_initv_ls
ds_reg_lbl_ls	ds_reg_maxv_lr
ds_reg_maxv_ls	ds_reg_minv_lr
ds_reg_minv_ls	ds_reg_mnem_ls
ds_reg_name_ls	ds_reg_radix_li
ds_reg_typ_li	ds_reg_typ_ls
ds_reg_typ_name_ls	ds_reg_unit_ls
ds_reg_wsize_li	ds_usr_par_ext_ls
ds_usr_par_ls	num_ds_in_i
num_ds_out_i	num_ds_usr_par_i
num_reg_in_ds_i	—

Table B-4. DocumentIt Block Parameters

Parameter	Parameter
blk_cmt_s	blk_cmt_ext_s
blk_code_cmt_s	blk_container_s
blk_derived_id_i	blk_fld_data_ls
blk_fld_keyname_ls	blk_has_in_b
blk_has_out_b	blk_has_outdata_b
blk_has_pars_b	blk_has_state_b
blk_id_i	blk_in_accu_lr
blk_in_accu_ls	blk_in_dsc_ls
blk_in_id_ls	blk_in_maxv_lr
blk_in_maxv_ls	blk_in_minv_lr
blk_in_minv_ls	blk_in_mnem_ls
blk_in_name_ls	blk_in_radix_li
blk_in_srcchn_li	blk_in_srcid_li
blk_in_typ_li	blk_in_typ_ls
blk_in_typ_name_ls	blk_in_unit_ls
blk_in_wsize_li	blk_keyname_s
blk_name_s	blk_out_accu_lr
blk_out_accu_ls	blk_out_dsc_ls
blk_out_id_ls	blk_out_maxv_lr
blk_out_maxv_ls	blk_out_memaddr_ls
blk_out_minv_lr	blk_out_minv_ls
blk_out_mnem_ls	blk_out_name_ls
blk_out_radix_li	blk_out_scope_ls
blk_out_typ_li	blk_out_typ_ls
blk_out_typ_name_ls	blk_out_unit_ls
blk_out_wsize_li	blk_par_enbl_lb

Table B-4. DocumentIt Block Parameters (Continued)

Parameter	Parameter
blk_par_val_ls	blk_par_vars_ls
blk_par_vars_s	blk_par_vars_sysidx_li
blk_par_vars_val_ls	blk_parent_id_i
blk_state_cmt_ls	blk_state_name_ls
blk_state_typ_li	blk_typ_s
blk_usr_par_ext_ls	blk_usr_par_ls
is_blk_b	is_ds_b
is_sb_b	is_std_b
is_text_b	num_blk_flds_i
num_blk_in_i	num_blk_out_i
num_blk_parvars_i	num_blk_state_i
num_blk_usr_par_i	—

Table B-5. DocumentIt STD Block Parameters

Parameter	Parameter
actv_cond_ls	bbl_cmt_ext_ls
bbl_cmt_ls	bbl_id_li
bbl_keyname_ls	bbl_name_ls
bbl_trn_off_li	bbl_typ_ls
is_blk_b	is_ds_b
is_sb_b	is_std_b
mealy_out_dsc_ls	moore_out_dsc_ls
num_bbl_trn_in_li	num_bbl_trns_li
num_bbls_in_std_i	num_mealy_out_li
num_moore_out_li	std_init_bbl_i
to_bbl_li	trn_cmt_ext_ls

Table B-5. DocumentIt STD Block Parameters (Continued)

Parameter	Parameter
trn_cmt_ls	trn_has_out_lb
trn_name_ls	trn_priority_li

Table B-6. DocumentIt Library Functions

Parameter	Parameter
include_img( )	large_frame( )
small_frame( )	—

C

Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Online technical support resources at ni.com/support include the following:
  - **Self-Help Resources**—For immediate answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
  - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Developer Exchange at ni.com/exchange. National Instruments Application Engineers make sure every question receives an answer.
- **Training and Certification**—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, NI Alliance Program members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Y