

MATLAB EXPO 2019

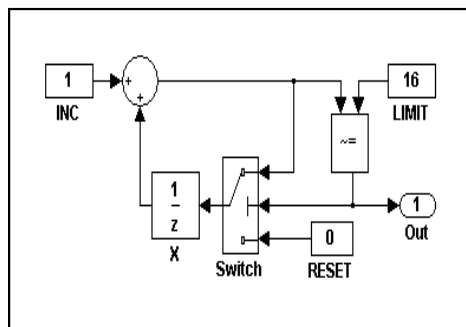
간편해진 C/C++ 코드 생성방법 소개

유재흥



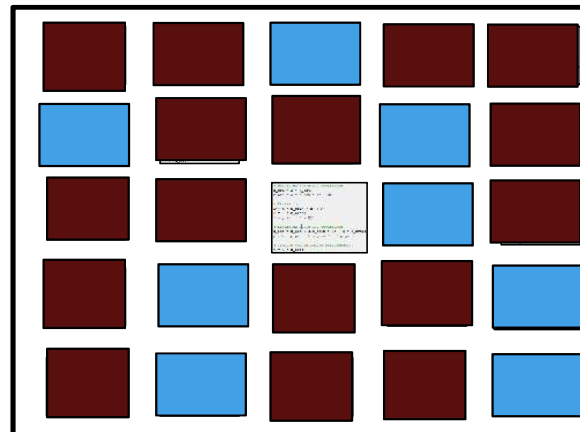
Goal: Generating Perfect Code for Your Environment

Model



Code
Generation

Application Framework



- Function Interfaces
- Data Definition
- Data Access

- How to do these?
- Any easy way to do these?



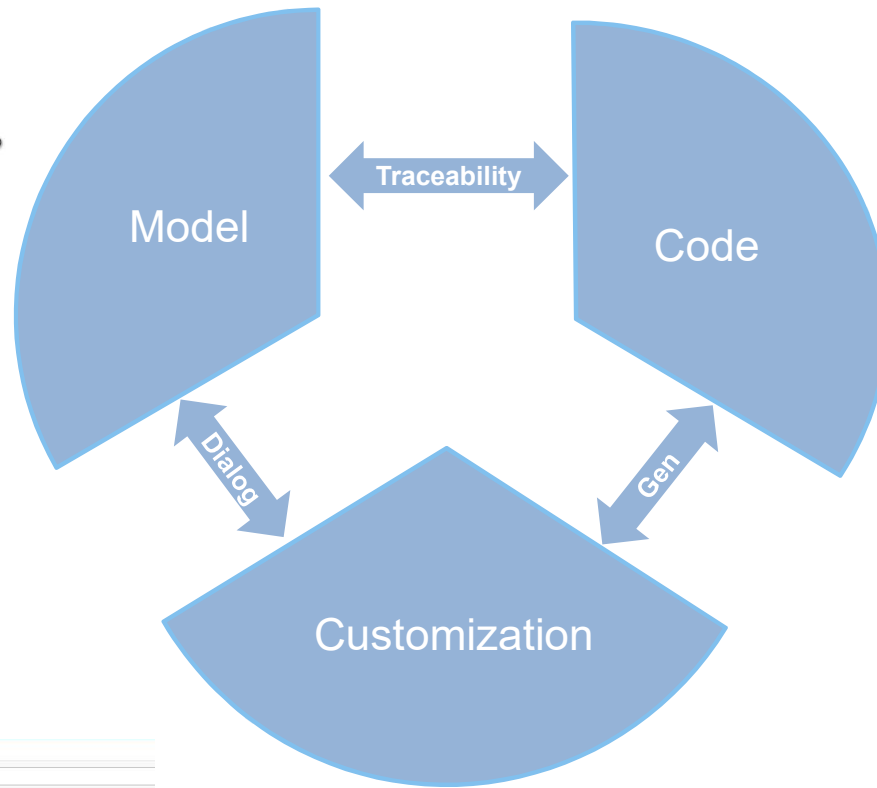
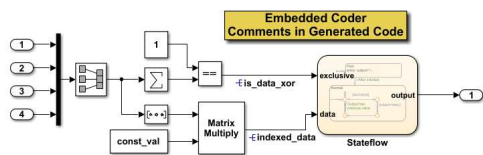
Agenda

- A Brand New Way to think about Customization
- A Brand New Way to interact with the Code
- Data Access Customization
- Row Major and Multi-Dimensional Indexing

Agenda

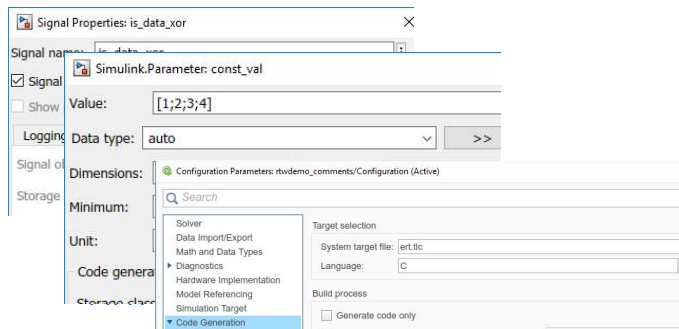
- A Brand New Way to think about Customization
- A Brand New Way to interact with the Code
- Data Access Customization
- Row Major and Multi-Dimensional Indexing

Code Customization Workflow (Before R2018A)

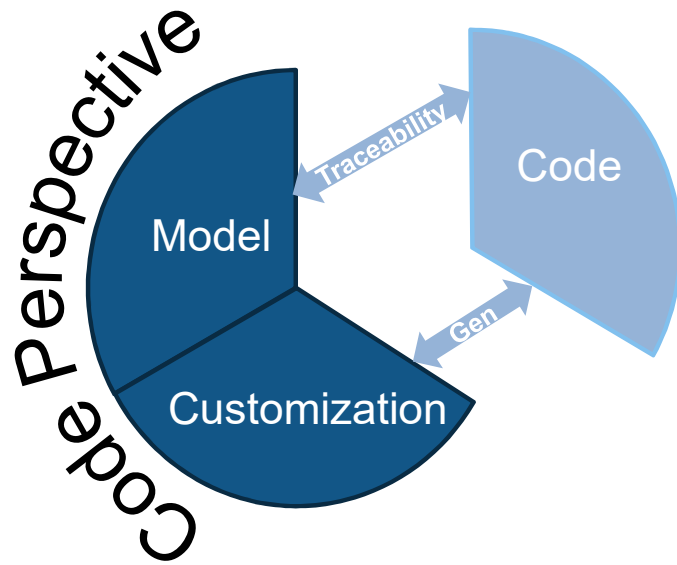


File: rtwdemo_comments.c

```
1 /******  
2 /* Prerelease License - for engineering feedback and testing purposes */  
3 /* only. Not for sale. */  
4 /*  
5 /* File: rtwdemo_comments.c */  
6 /*  
7 /* Code generated for Simulink model 'rtwdemo_comments'. */  
8 /*  
9 /* Model version : 1.212 */  
10 /* Simulink Coder version : 9.0 (R2018b Prerelease) 19-Dec-2017 */  
11 /* TIC version : 9.0 (Dec 19 2017) */  
12 /* C/C++ source code generated on : Sun Jan 7 21:23:06 2018 */  
13 /*  
14 /* You can customize this banner by specifying a different template. */  
15 /* MODEL DESCRIPTION:  
16 /* User-Controlled Comments to Improve Code Readability and Traceability */  
17 /*  
18 /* This example shows how to add comments to numerous types  
19 /* of objects in Simulink(R) and Stateflow(R).  
20 /* Description added in Stateflow note  
21 /*  
22 /* HISTORY:  
23 /* History added in doc block.  
24 /*  
25 /* ABSTRACT:  
26 /* Abstract added in annotation  
27 /******
```



Code Customization Workflow with Code Perspective (R2018a)



Model Description

This model represents a mutual exclusion arbitrator. If exactly one input is enabled, the output will be its input. Otherwise, the previous output will be used.

Comments in Generated Code

Adding comments to your model is an easy way of creating traceability between the model and generated code. The two steps are:

- 1) Add comments to model.
- 2) Check configuration parameter options.

Step 1: Add Comments to Model

Comments can be added to numerous types of objects in Simulink and Stateflow, including the following (double-click to open):

- **sa templates**
- St
- St
- St
- St
- M
- M
- S:al

Step 2: Check Comments Options

Open the model's configuration parameters and navigate to the Comments section of the Code Generation settings, or double-click below to see these settings.

Model Data Editor

Source	Name	Resolve	Storage Class	Header File	Definition File	Get Function	Set Function	Struct Name
In4		<input type="checkbox"/>	Auto					
Mux		<input type="checkbox"/>	Auto					
One selected	is_data_xor	<input checked="" type="checkbox"/>						
Product	indexed_data	<input checked="" type="checkbox"/>						
Reshape		<input type="checkbox"/>	Auto					

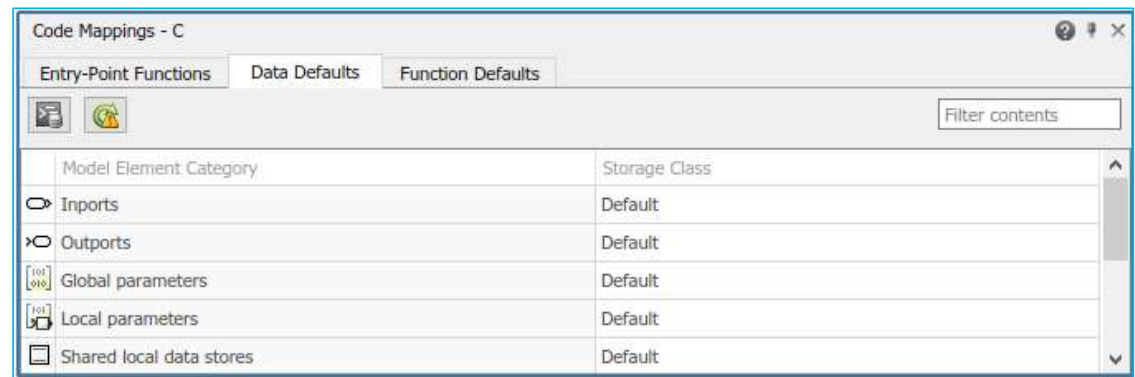
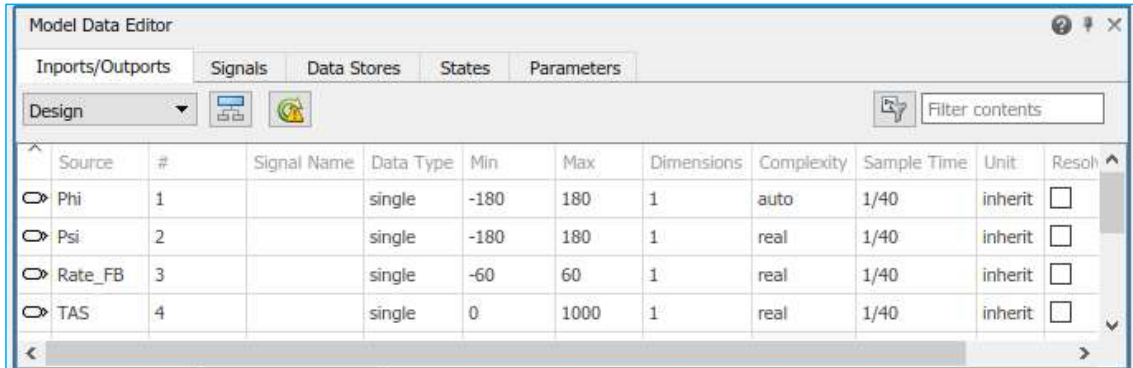
Overview of Code Inspector

The screenshot displays the MATLAB Code Inspector interface for a Simulink model named 'rtwdemo_comments'. The main workspace shows a Simulink block diagram with a yellow box labeled 'Embedded Code Comments in Generated Code' and a blue button 'Generate Code Using Embedded Code (double-click)'. Below the diagram, there are three instructional panels: 'Description', 'Step 1: Add Comments to Model', and 'Step 2: Check Comments Options'. The 'Code Mappings - C' pane at the bottom shows a table of model element categories and their storage classes.

Model Element Category	Storage Class
Inputs	Default
Outputs	Default
Global parameters	Default
Local parameters	Default
Shared local data stores	Default

Customization Spreadsheets

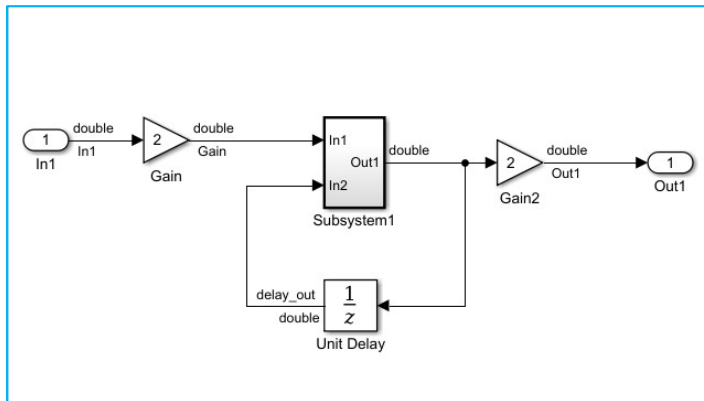
- Code Mappings allow for default specification of functions and variables
- Model Data Editor allows for individual element customization

The screenshot shows the 'Model Data Editor' window with the 'Inports/Outports' tab selected. It displays a table of signal parameters for four sources: Phi, Psi, Rate_FB, and TAS.

Source	#	Signal Name	Data Type	Min	Max	Dimensions	Complexity	Sample Time	Unit	Resolv
Phi	1		single	-180	180	1	auto	1/40	inherit	<input type="checkbox"/>
Psi	2		single	-180	180	1	real	1/40	inherit	<input type="checkbox"/>
Rate_FB	3		single	-60	60	1	real	1/40	inherit	<input type="checkbox"/>
TAS	4		single	0	1000	1	real	1/40	inherit	<input type="checkbox"/>

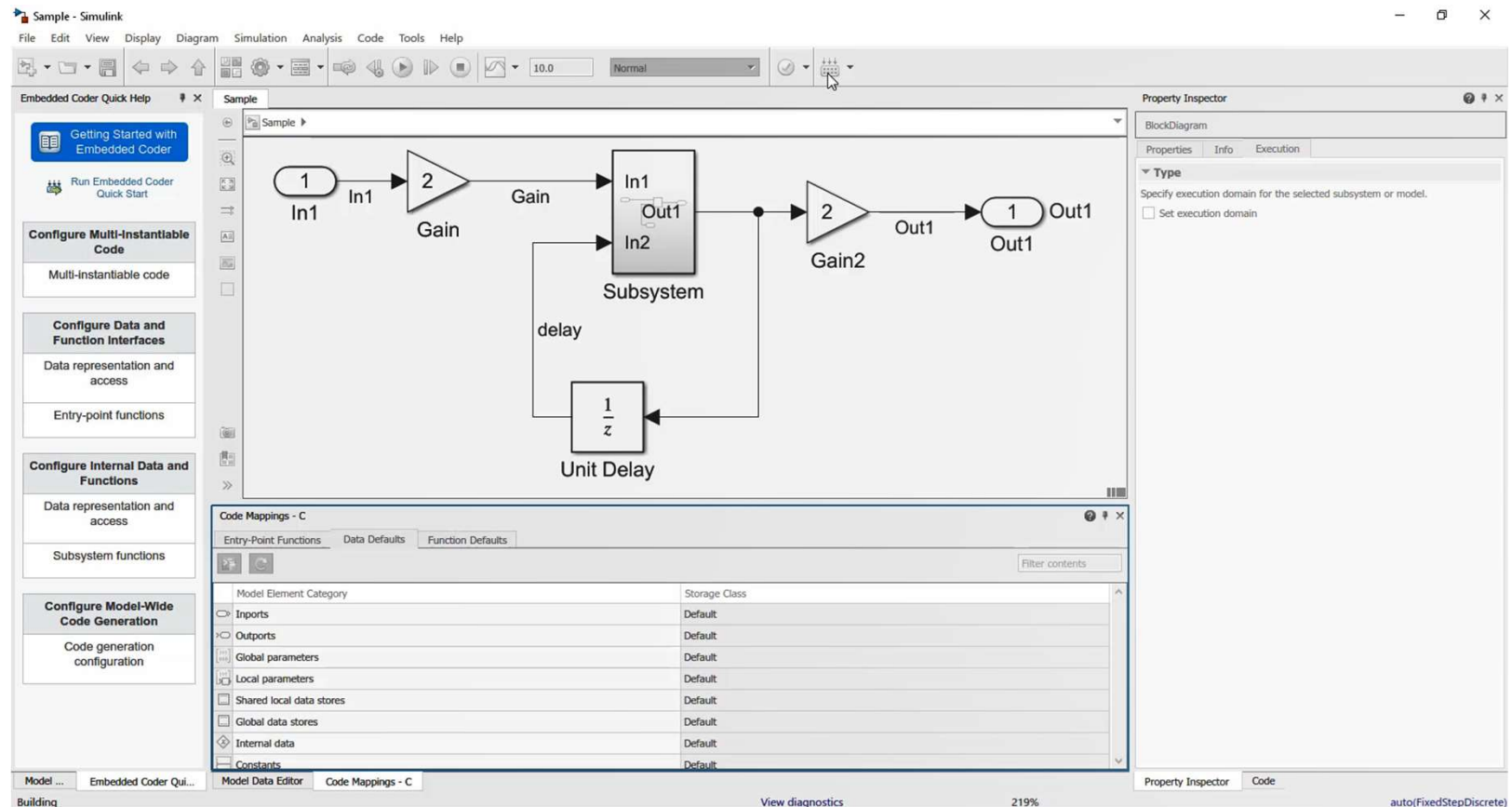
A Case Study



```
extern real_T *In1;
real_T Out1;
static real_T delay;
```

Code Mappings	
Data Defaults	Function Defaults
Filter contents	
Model Element Category	Storage Class
Inports	ImportedExternPointer
Outports	ExportedGlobal
Global parameters	Default
Constants	Default
Internal data	FileScope

Generating code with C Code mapping



The screenshot displays the MATLAB/Simulink environment. The main workspace shows a block diagram of a control system. The diagram includes an input port 'In1' (represented by a circle with '1'), a gain block 'Gain' (represented by a triangle with '2'), a subsystem block 'Subsystem' (represented by a rectangle with 'In1', 'In2', and 'Out1' ports), a unit delay block 'Unit Delay' (represented by a rectangle with '1/z'), and an output port 'Out1' (represented by a circle with '1'). The signal flow is: In1 → Gain → Subsystem → Gain2 → Out1. A feedback path goes from the output of Gain2 through the Unit Delay block back to the 'In2' input of the Subsystem block.

Below the block diagram, the 'Code Mappings - C' window is open, showing the 'Entry-Point Functions' tab. This window lists various model elements and their corresponding storage classes for C code generation.

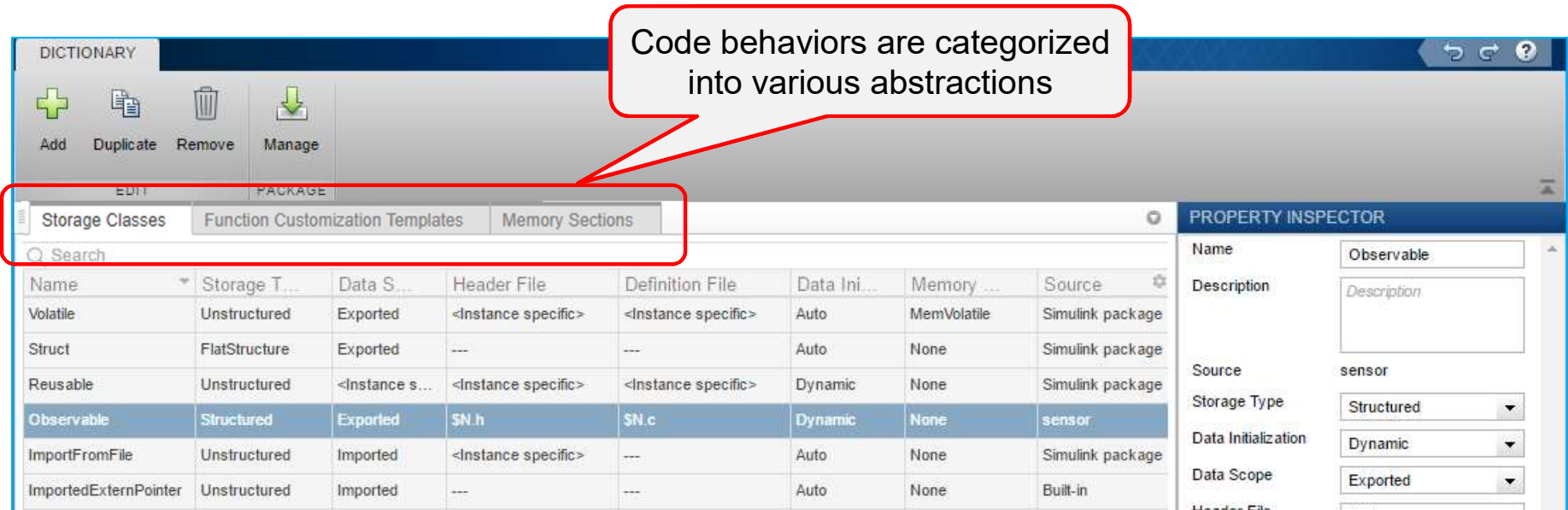
Model Element Category	Storage Class
Inputs	Default
Outputs	Default
Global parameters	Default
Local parameters	Default
Shared local data stores	Default
Global data stores	Default
Internal data	Default
Constants	Default

The right side of the interface shows the 'Property Inspector' window, which is currently displaying the 'BlockDiagram' tab. The 'Type' section is visible, with a checkbox for 'Set execution domain'.

Defining Code Generation Behavior Without Packages and Classes

- Embedded Coder Dictionary

Code behaviors are categorized into various abstractions



Name	Storage T...	Data S...	Header File	Definition File	Data Ini...	Memory ...	Source
Volatile	Unstructured	Exported	<Instance specific>	<Instance specific>	Auto	MemVolatile	Simulink package
Struct	FlatStructure	Exported	---	---	Auto	None	Simulink package
Reusable	Unstructured	<Instance s...	<Instance specific>	<Instance specific>	Dynamic	None	Simulink package
Observable	Structured	Exported	\$N.h	\$N.c	Dynamic	None	sensor
ImportFromFile	Unstructured	Imported	<Instance specific>	---	Auto	None	Simulink package
ImportedExternPointer	Unstructured	Imported	---	---	Auto	None	Built-in

PROPERTY INSPECTOR

Name: Observable

Description: Description

Source: sensor

Storage Type: Structured

Data Initialization: Dynamic

Data Scope: Exported

Header File: ...

Step to Set Embedded Coder Dictionary

Storage Classes
Function Customization Template
Memory Sections

Name	Comment	Pre Statement	Post Statement	
MemConst	/* Const memory section */			Simulink package
MemVolatile	/* Volatile memory section */			Simulink package
MemConstVolatile	/* ConstVolatile memory s...			Simulink package
MyMemorySection		#pragma section near=...	#pragma endsection	Each variable

Setting Memory Sections

MyMemorySection

Description

Sample

Comment

Pre Statement

Post Statement

Statements Surround

PSEUDOCODE PREVIEW

Declaration:

```
#pragma section near=RAM_ASW
extern DATATYPE DATANAME;
#pragma endsection
```

Definition:

```
#pragma section near=RAM_ASW
DATATYPE DATANAME;
#pragma endsection
```

Step to Set Embedded Coder Dictionary

Storage Classes

Function Customization Template

Memory Sections

Search

Name	Comment	Pre Statement	Post Statement		
MemConst	/* Const memory section */				Simulink package
MemVolatile	/* Volatile memory section */				Simulink package
MemConstVolatile	/* ConstVolatile memory s...				Simulink package
MyMemorySection		#pragma section near=...	#pragma endsection	Each variable	Sample

Setting Memory Sections

MyMemorySection

Description

Sample

Source

Comment

Pre Statement

Post Statement

Statements Surround

#pragma section near=RAM_ASW

#pragma endsection

Each variable

PSEUDOCODE PREVIEW

Declaration:

```
#pragma section near=RAM_ASW
extern DATATYPE DATANAME;
#pragma endsection
```

Definition:

```
#pragma section near=RAM_ASW
DATATYPE DATANAME;
#pragma endsection
```

Rich code preview to assist in defining customizations

Step to Set Embedded Coder Dictionary

Storage Classes

Function Customization Templates

Memory Sections

Name	Data Scope	Header File	Storage	Access Function	Macro	Dynamic	Simulink package
FileScope	File	---	Unstructured	Auto	None	---	Simulink package
Localizable	Auto	---	Unstructured	Auto	None	---	Simulink package
Struct	Exported	---	FlatStructure	Auto	None	---	Simulink package
GetSet	Imported	<Instance specific>	AccessFunction	Auto	---	---	Simulink package
CompilerFlag	Imported	---	Unstructured	Macro	---	---	Simulink package
Reusable	<Instance spe...	<Instance specific>	Unstructured	Dynamic	None	---	Simulink package
SignalStruct	Exported	SN.h	Structured	Dynamic	None	---	Sample
ParamStruct	Exported	SN.h	Structured	Static	None	---	Sample
MyStorage	Exported	SN.h	Unstructured	Dynamic	MyMemorySection	---	Sample

Setting for Variables & Functions

PROPERTY INSPECTOR

Name:

Description:

Source:

File Placement

Data Scope:

Header File:

Definition File:

Storage

Storage Type:

Data Initialization:

Memory Section:

Qualifiers

Allowed Usage

☐ Parameters
☒ Signals

PSEUDOCODE PREVIEW

For single-instance data:

Declaration: exported through file: "Sample.h"

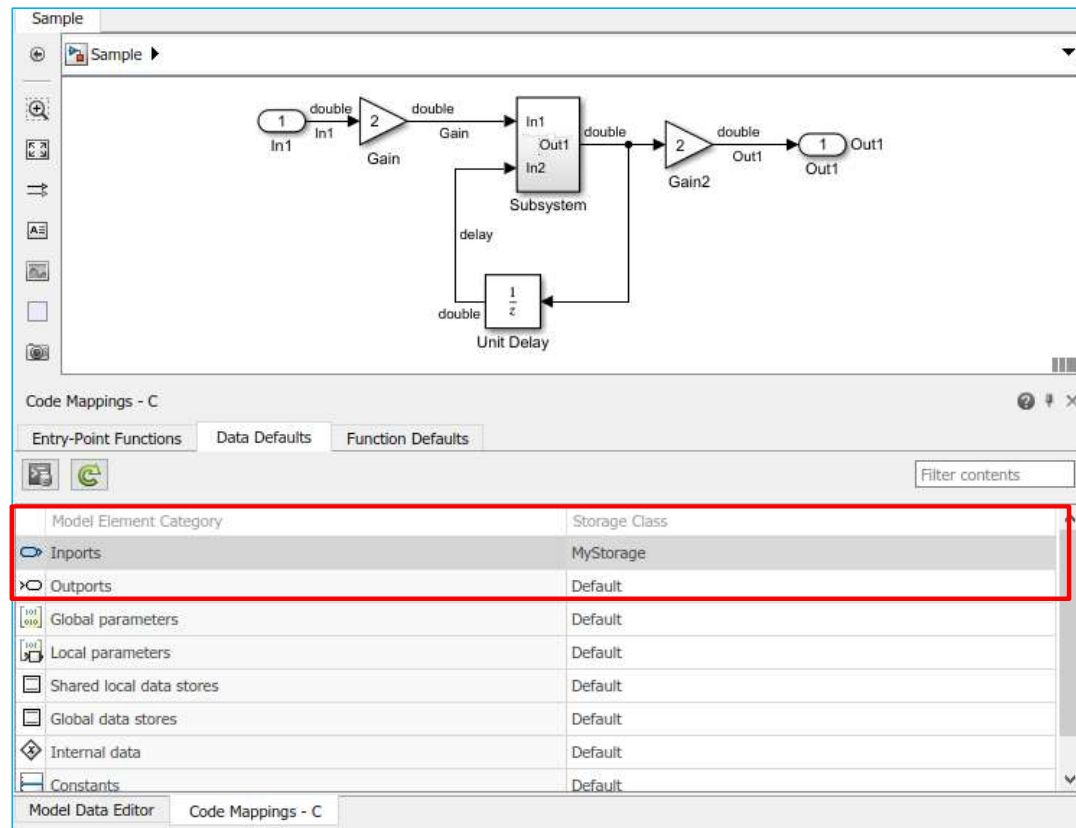
```
#pragma section near=RAM_ASIM
extern DATATYPE DATANAME;
#pragma endsection
```

Definition: defined in file: "Sample.c"

```
#pragma section near=RAM_ASIM
DATATYPE DATANAME;
#pragma endsection
```

For multi-instance data:

Loading Storage Class from Embedded Coder Dictionary



Configuring C Step function interface

Configure the generated C Step function interface, including function name and arguments.

C function prototype: `arg_Ail_Cmd = rtwdemo_roll_step(* arg_Phi, arg_Psi, arg_Rate_FB, arg_...`

C Step Function Name:

☒ Configure arguments for Step function prototype

(* invokes update diagram)

C return argument:

Port Name	Port Type	C Type Qualifier	C Identifier Name
Phi	Inport	Pointer	arg_Phi
Psi	Inport	Value	arg_Psi
Rate_FB	Inport	Value	arg_Rate_FB
TAS	Inport	Value	arg_TAS
AP_Eng	Inport	Value	arg_AP_Eng
HDG_Mode	Inport	Value	arg_HDG_Mode

Roll Axis Autopilot Model

Code

```
rtwdemo_roll.c
/* Block signals and states (default storage) */
DW rtDW;

/* Model step function */
real32_T rtwdemo_roll_step(real32_T *arg_Phi, real32_T arg_Psi, real32_T
arg_Rate_FB, real32_T arg_TAS, boolean_T arg_AP_Eng, boolean_T arg_HDG_Mode,
real32_T arg_HDG_Ref, real32_T arg_Turn_Knob)

boolean_T rtb_NotEngaged_f;
real32_T rtb_Sum1;

/* specified return value */
real32_T arg_Ail_Cmd;

/* Outputs for Atomic SubSystem: '<Root>/RollAngleReference' */
/* UnitDelay: '<S7>/FixPt Unit Delay1' */
rtb_Sum1 = rtDW.FixPtUnitDelay1_DSTATE;

/* Switch: '<S7>/Enable' incorporates:
* Input: '<Root>/AP_Eng'
* Logic: '<S3>/NotEngaged'
*/
if (arg_AP_Eng) {
```

Function Name

rtwdemo_roll_initialize

rtwdemo_roll_step

Configure Prototype

Generating C-Code with C Step function interface

Configure C Step Function Interface: rtwdemo_roll

Configure the generated C Step function interface, including function name and arguments.

C function prototype: `arg_Ail_Cmd = MyFunction(* arg_Phi, arg_Psi, arg_Rate_FB...`

C Step Function Name:

☒ Configure arguments for Step function prototype

(* invokes update diagram)

C return argument:

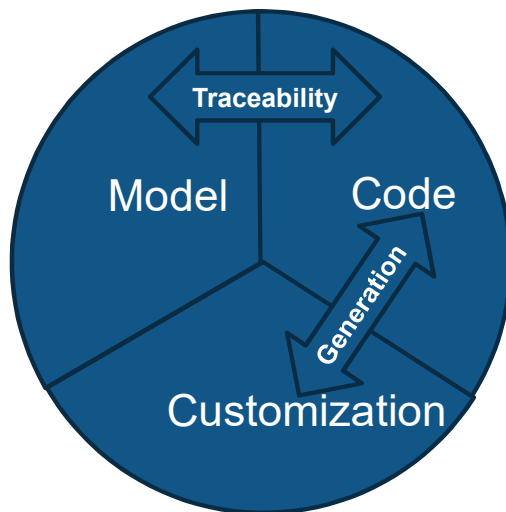
Port Name	Port Type	Pointer	arg_Phi
Psi	Input	Value	arg_Psi
Rate_FB	Input	Value	arg_Rate_FB
TAS	Input	Value	arg_TAS
AP_Eng	Input	Value	arg_AP_Eng
HDG_Mode	Input	Value	arg_HDG_Mode
HDG Ref	Input	Value	arg_HDG Ref

OK Cancel Help Apply

Agenda

- A Brand New Way to think about Customization
- A Brand New Way to interact with the Code
- Data Access Customization
- Row Major and Multi-Dimensional Indexing

In R2018b we will integrate the last piece, the code



The screenshot shows the MATLAB R2018b interface with the 'rtwdemo_comments' model open. The model diagram on the left shows a block with inputs 1, 2, 3, and 4. The 'Code' window on the right shows the generated C code for 'rtwdemo_comments.c'. The code includes comments for inputs and a block description for 'Sum'. The 'Code Mappings' window at the bottom shows the mapping between the model's entry-point functions and the generated code functions.

Model Description

This model represents a mutual exclusion arbitrator. If exactly one input is enabled, the output will be its input. Otherwise, the previous output will be used.

Comments in Generated Code

Adding comments to your model is an easy way of creating traceability between the model and generated code. The two steps are:

- 1) Add comments to model.
- 2) Check configuration parameter options.

Step 1: Add Comments to Model

Comments can be added to numerous types of objects in Simulink and Stateflow, including the following (double-click to open):

- Simulink Block
- Stateflow State
- Simulink Signal Object
- Simulink Parameter Object
- MPT Signal Object
- MPT Callback Function

Step 2: Check Comments Options

Open the model's configuration parameters and navigate to the Comments section of the Code Generation settings, or double-click below to see these settings.

Additional Documentation

Additional documentation is available for integrating model comments into generated code by double-clicking the link below.

Customize Banners

Customize file banner, function banner, or file trailer by double-clicking the link below.

Code Mappings - C

Source	Function Customization Template	Function Name
fx Initialize Function	Model default: Default	rtwdemo_comments_initialize
fx Step Function [Sample Time:0.2s]	Model default: Default	rtwdemo_comments_step

Showing Traceability Model to Code

Embedded Code Comments in Generated Code

Model Description

This model represents a mutual exclusion arbitrator. If exactly one input is enabled, the output will be its input. Otherwise, the previous output will be used.

Comments in Generated Code

Adding comments to your model is an easy way of creating traceability between the model and generated code. The two steps are:

Step 1: Add Comments to Model

Comments can be added to numerous types of objects in Simulink and Stateflow, including the following (double-click to open):

- Simulink Block
- Stateflow State
- Stateflow Transition
- Simulink Signal Object
- Simulink Parameter Object

Step 2: Check Comments

Open the model's configuration and navigate to the Code Generation settings double-click below to see

Additional Documentation

Additional documentation for integrating model comments into generated code by default.

Code Mappings - C

Source	Function Customization Template	Function Name
fx Initialize Function	Model default: Default	rtwdemo_comments_initialize
fx Step Function [Sample Time:0.2s]	Model default: Default	rtwdemo_comments_step

Code

```

rtwdemo_comments.c (9)
Highlighting: <Root>/Stateflow
Show comments and empty lines
Open code generation report

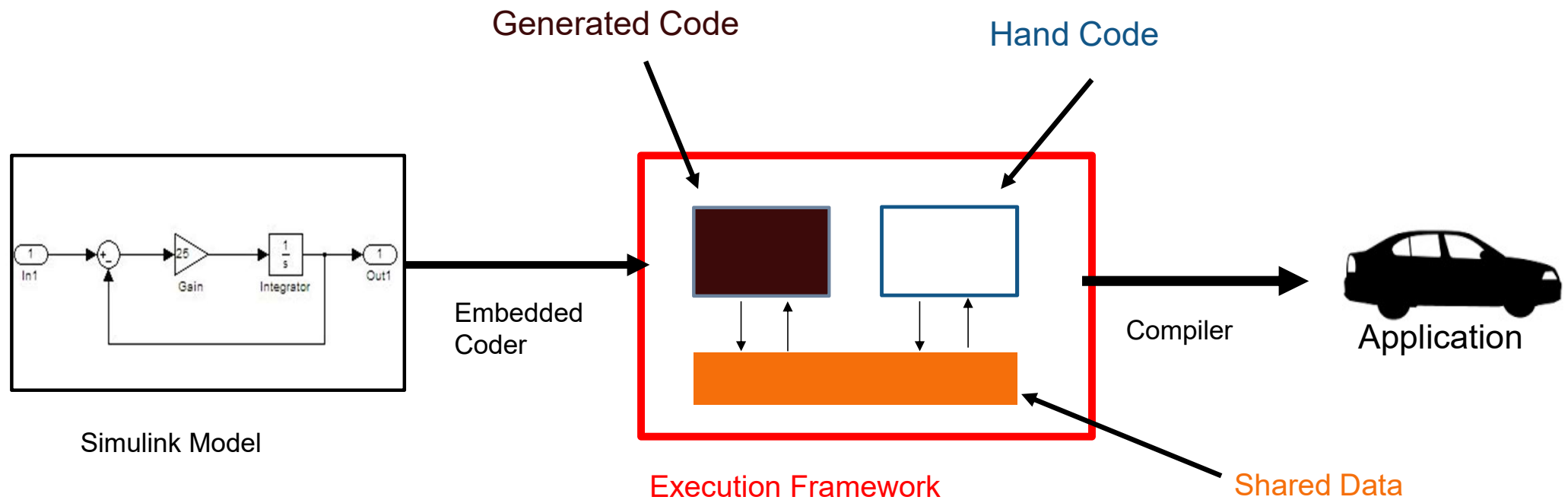
92  /* Block description for '<Root>/Product':
93  /* This block determines the output index by
94  /* the vector of inputs and [1 2 3 4].
95  */
96  indexed_data = ((rtU.In1 * const_val[0] + rtU.In2 * const_val[1]) + rtU.In3
97  const_val[2]) + rtU.In4 * const_val[3];
98
99
100 /* Chart: '<Root>/Stateflow'
101 /*
102 /* Block description for '<Root>/Stateflow':
103 /* Stateflow block decides the index output by following logic
104 /* 1. If one and only one input signal is enabled, the index will
105 /* refer to the enabled signal line.
106 /* 2. Otherwise, maintain the old value.
107 /*
108 /* Gateway: Stateflow */
109 /* During: Stateflow */
110 if (rtDWork.bitsForTID0.is_active_c1_rtwdemo_comments == 0U) {
111 /* Entry: Stateflow */
112 rtDWork.bitsForTID0.is_active_c1_rtwdemo_comments = 1;
113
114 /* Entry Internal: Stateflow */
115 /* Transition: '<S2>:9' */
116 rtDWork.bitsForTID0.is_c1_rtwdemo_comments = IN_First;
117
118 /* Output: '<Root>/Out1' */
119 /* Entry 'First': '<S2>:2' */
120 rtY.Out1 = 1;
121 } else if (rtDWork.bitsForTID0.is_c1_rtwdemo_comments == IN_First) {

```

Agenda

- A Brand New Way to think about Customization
- A Brand New Way to interact with the Code
- **Data Access Customization**
- Row Major and Multi-Dimensional Indexing

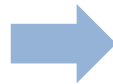
Deployment Scenario: Generated Code as Part of Application



Control *how* data is accessed

- Separate the algorithm code from how data is stored and accessed

```
void model_step()
{
    real_T rtb_Gain;
    rtb_Gain = 2.0 * ExtU.Temperature;
}
```



```
void model_step()
{
    real_T rtb_Gain;
    rtb_Gain = 2.0 * getTemperature();
}
```

Generated Code

```
real_T getTemperature()
{
    ensureTemperatureDataCorrectness();
    return temperature;
}
```

User Code

Data Access Customization Capabilities in R2019a

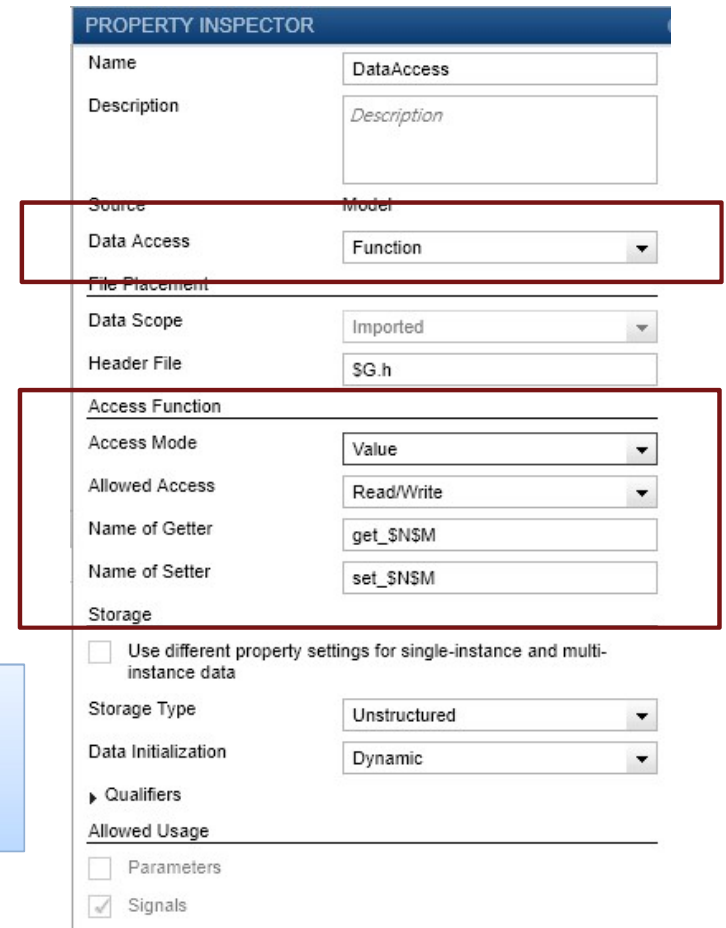
- Applicable to Root I/O and Parameters
- Improvements over current Get/Set
 - Export with header file and definition file
 - Richer function name specification
 - Pass by value vs. Pass by pointer

```
real_T getVector(int elem)
{
    return data[elem];
}
```

By value

```
real_T* getVector()
{
    return data;
}
```

By pointer



PROPERTY INSPECTOR

Name:

Description:

Source:

Data Access:

File Placement

Data Scope:

Header File:

Access Function

Access Mode:

Allowed Access:

Name of Getter:

Name of Setter:

Storage

☐ Use different property settings for single-instance and multi-instance data

Storage Type:

Data Initialization:

Qualifiers

Allowed Usage

☐ Parameters

☒ Signals

Agenda

- A Brand New Way to think about Customization
- A Brand New Way to interact with the Code
- Data Access Customization
- Row Major and Multi-Dimensional Indexing

Row Major vs. Column Major

Column major
code
generation:

$M[] = \{11, 21, 31, 12, 22, 32\};$

$M(2) = 31$

Column-major
indexing

MATLAB:

$M = \begin{bmatrix} 11 & 12 \\ 21 & 22 \\ 31 & 32 \end{bmatrix}$



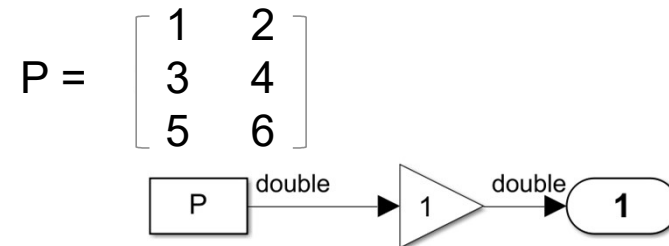
Row major
code
generation:

$M[] = \{11, 12, 21, 22, 31, 32\};$

$M(2) = \mathbf{21}$

Row-major
indexing

Row Major Code Generation



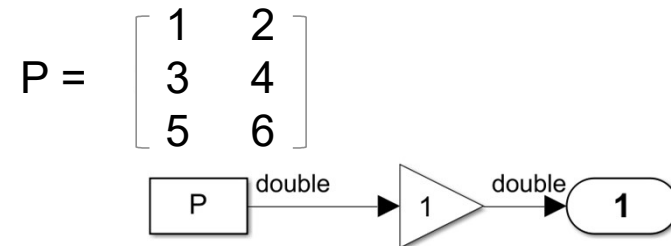
Column major layout

```
P rtP = {
  /* Variable: P
   * Referenced by: '<Root>/Constant'
   */
  { 1.0, 3.0, 5.0, 2.0, 4.0, 6.0 }
};
```

Row major layout

```
P rtP = {
  /* Variable: P
   * Referenced by: '<Root>/Constant'
   */
  { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 }
};
```

Row Major and Multidimension Indexing



Row major layout

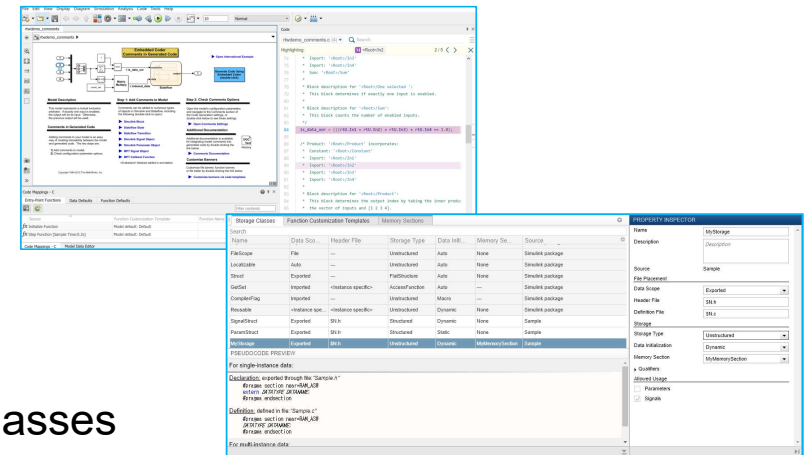
Multi-Dimensional layout

```
P rtP = {
    /* Variable: P
    * Referenced by: '<Root>/Constant'
    */
    { 1.0, 3.0, 5.0, 2.0, 4.0, 6.0 }
};
```

```
P[3][2] = { { 1.0, 2.0 }, { 3.0, 4.0 }, { 5.0, 6.0 } };
```

Key Takeaways

- Code Perspective
 - Enable to set Default Code Pattern of Blocks
 - Modifying Block Properties in Spreadsheets
 - Showing Model and Code in same Window
- Embedded Code Dictionary
 - Defining Code Behavior without Package and Classes
- Data Access Customization
 - Accessing Data with Function
- Row Major Layout and Multi-Dimensional Indexing
 - Supporting Row Major and Multi-Dimensional Indexing



```
void model_step()
{
    real T rtb_Gain;
    rtb_Gain = 2.0 * ExtU.Temperature;
}
```

➔

```
void model_step()
{
    real T rtb_Gain;
    rtb_Gain = 2.0 * getTemperature();
}
```

Generated Code

```
real T getTemperature()
{
    ensureTemperatureDataCorrectness();
    return temperature;
}
```

User Code

Row major layout

Multi-Dimensional layout

```
P.rtp = {
    /* Variable: P
    * Referenced by: '(Host)/Constant'
    */
    { 1.0, 3.0, 5.0, 2.0, 4.0, 6.0 }
};
```

```
P[3][2] = { { 1.0, 2.0 }, { 3.0, 4.0 }, { 5.0, 6.0 } };
```

MATLAB EXPO 2019

감사합니다

