**Embedded Coder Dictionary**

<https://www.mathworks.com/help/releases/R2020a/ecoder/ref/embeddedcoderdictionary.html>

<https://translate.google.co.jp/translate?hl=ja&tab=rT&sl=en&tl=ja&u=https%3A%2F%2Fde.mathworks.com%2Fhelp%2Freleases%2FR2020a%2Fecoder%2Fref%2Fembeddedcoderdictionary.html>

モデルデータと関数のコード生成を制御するコード定義を作成する

[ページ全体を展開](javascript:void(0);)

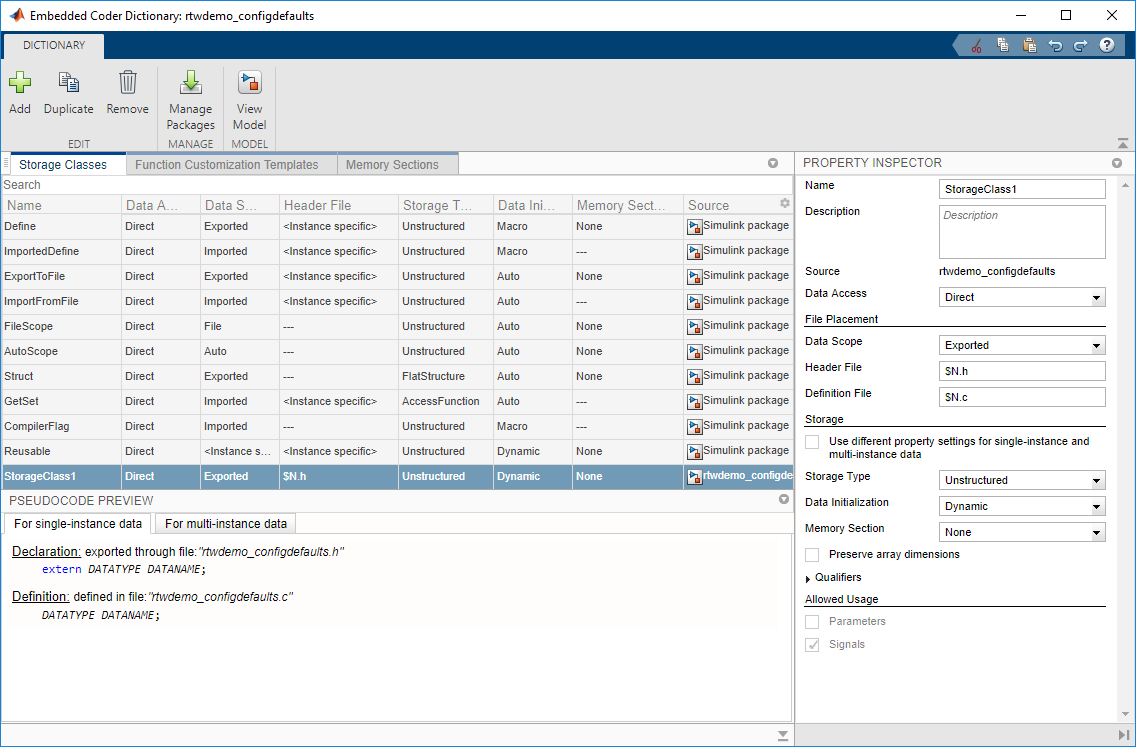
**説明**

Embedded Coder Dictionaryは、カスタムコード定義を作成するためのグラフィカルインターフェイスです。これらの定義をモデルに適用することにより、デフォルトで特定のソフトウェアアーキテクチャに準拠するコードを生成できます。たとえば、独自のストレージクラスを作成して、デフォルトでルートレベルの入力などのモデルデータのカテゴリに適用することができます。

次のタイプのコード定義を作成できます。

* モデルデータ用に生成されたコードを制御するストレージクラス。
* などのモデルエントリポイント関数の名前を制御する関数カスタマイズテンプレート *model*\_step。テンプレートはまた、エントリポイント関数にメモリセクションを適用します。
* メモリ内のデータと関数の配置を制御するメモリセクション。生成されたコードには、構文を指定するプラグマなどのカスタム装飾が含まれます。

コード生成定義の作成に関する一般的な情報については、「[Define Storage Classes、Memory Sections、and Function Templates for Software Architecture」を](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ug/define-storage-class-memory-section-and-function-template-settings-for-a-software-architecture.html&usg=ALkJrhjZ24TZTUcg062CLOaujh22tfnz1w)参照してください。



**もっと**

**Embedded Coderディクショナリを開く**

* Embedded Coderディクショナリを開くには、次のいずれかの方法を使用します。
  + コードマッピングエディター（「[コードマッピングエディター](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ref/codemappingseditor.html&usg=ALkJrhiy2gG9isGNGy4yhBTJ4xdjnYkqSA)」を参照）で、[埋め込みCoder辞書]アイコンをクリックします。https://de.mathworks.com/help/releases/R2020a/ecoder/ref/icon_code_dictionary.png。
  + モデルウィンドウで、上の**Cコード**タブ、選択し **[設定]** > [ **組み込みCoderの辞書を**。

Embedded Coder Dictionaryウィンドウには、モデルファイルに保存されているコード生成定義が表示されます。モデルがデータディクショナリにリンクされている場合、ウィンドウにはそのデータディクショナリ、または該当する場合は参照されているディクショナリに格納されている定義も表示されます。「**ソース」**列は、各定義が保管されている場所を示しています。

* Simulinkで埋め込まれたCoderの辞書を開くには®モデルエクスプローラでは、データ・ディクショナリ**モデルの階層構造** ]ペイン：
  + 辞書ノードの下で、**Embedded Coder**ノードを選択します。

ノードが表示されない場合は、ディクショナリノードを右クリックして、[ **空のセクションの表示** ]を選択します。

* + ダイアログペイン（右側のペイン）で、[ **埋め込みCoder辞書を開く** ]をクリックします。

**例**

**ストレージクラスの作成と確認**

モデルで、ブロック状態を含む内部モデルデータを、特性を制御できる構造に集約するストレージクラスを作成します。次に、モデルからコードを生成して、ストレージクラスを確認します。

1. サンプルモデルを開きrtwdemo\_rollます。

rtwdemo\_roll

1. コードパースペクティブでモデルが開かない場合は、Embedded Coderアプリを開き、[ **Cコード** ]タブをクリックします。
2. モデルダイアグラムの下のコードマッピングエディターで、[ **データのデフォルト** ]タブを選択し ます。
3. Embedded Coder Dictionaryアイコンをクリックします https://de.mathworks.com/help/releases/R2020a/ecoder/ref/icon_code_dictionary.png。Embedded Coder Dictionaryウィンドウには、モデルファイルに保存されているコード生成定義が表示されます。
4. [Embedded Coder Dictionary]ウィンドウで、[ **追加** ] **を**クリックします 。
5. リストの下部に表示される新しいストレージクラスを選択し StorageClass1ます。右側の[プロパティインスペクター]ペインで、この表にリストされているプロパティ値を設定します。

| **物件** | **値** |
| --- | --- |
| **名前** | InternalStruct |
| **ヘッダーファイル** | internalData\_$R.h |
| **定義ファイル** | internalData\_$R.c |
| **保管タイプ** | Structured |
| **構造物プロパティ** > **タイプ名** | internalData\_T\_$M |
| **構造物プロパティ** > **インスタンス名** | internalData\_$M |

1. 変更を加えたら、下部のペインで、疑似コードのプレビューが期待どおりに反映されていることを確認します。
2. コードマッピングエディターに戻ります。**内部データ**行を選択し、**ストレージクラス**をに設定し InternalStructます。
3. [コンフィギュレーションパラメーター]ダイアログボックスの[ **コード生成]** > [ **コード配置]**ペインで、**[ファイルのパッケージ形式]**をに 設定し **ます**Modular。
4. コードを生成します。
5. Simulinkエディターのコードビューで、ファイルを開いて検査します internalData\_rtwdemo\_roll.h。ファイルは構造タイプを定義しinternalData\_T\_、そのフィールドはモデルのブロック状態を表します。
6. / \*ストレージクラス 'InternalStruct'、システム '<Root>' \* /
7. typedef struct {
8. real32\_T FixPtUnitDelay1\_DSTATE; / \* '<S7> / FixPt Unit Delay1' \* /
9. real32\_T Integrator\_DSTATE; / \* '<S1> /インテグレーター' \* /
10. int8\_T Integrator\_PrevResetState; / \* '<S1> /インテグレーター' \* /

} internalData\_T\_;

このファイルでは、という名前のグローバル構造変数も宣言しています internalData\_。

/ \*ストレージクラス 'InternalStruct' \* /

extern internalData\_T\_ internalData\_;

1. ファイルを開いて検査しますinternalData\_rtwdemo\_roll.c。ファイルはにメモリを割り当てますinternalData\_。
2. / \*ストレージクラス 'InternalStruct' \* /

internalData\_T\_ internalData\_;

**関数カスタマイズテンプレートの作成**

関数テンプレートを使用すると、生成されたエントリポイント関数の名前を管理するルールを指定できます。この手法は、エクスポート関数モデルやマルチレート、マルチタスクモデルなど、多くのエントリポイント関数を持つモデルで時間とメンテナンスの労力を節約するのに役立ちます。

この例は、命名規則を指定する関数テンプレートを作成する方法を示していますfunc\_$N\_$R。$Nは、生成された各関数のベース名で$Rあり、Simulinkモデルの名前です。

1. サンプルモデルを開きrtwdemo\_mrmtbbます。
2. ブロック図を更新します。このマルチタスクモデルには2つの実行率があるため、生成されたコードには2つの対応するエントリポイント関数が含まれます。
3. モデルで、モデル構成パラメーター**System target file**をに設定しますert.tlc。関数カスタマイズテンプレートを使用するには、ERTベースのシステムターゲットファイルを使用する必要があります。
4. Simulinkエディターで、コードパースペクティブモードを有効にして、Embedded Coder辞書を開きます。
5. Embedded Coderディクショナリの[ **関数のカスタマイズテンプレート** ]タブで、[ **追加** ]をクリックします。
6. 新しい関数テンプレートでは、次のプロパティを設定します。
   * **に名前**を付け myFunctionsます。
   * **関数名**をに func\_$N\_$R。

変更を行った後、疑似コードのプレビューが期待どおりに反映されていることを確認します。

1. 上の**機能のデフォルト**タブ、のために **初期化/終了**および **実行の**行セット**機能のカスタマイズテンプレート**へ myFunctions。
2. コードを生成します。
3. コードパースペクティブの右側のコードビューペインで、ファイルを開いて検査しますrtwdemo\_mrmtbb.c。ファイルには2つの実行関数を定義して、func\_step0\_rtwdemo\_mrmtbbそして func\_step1\_rtwdemo\_mrmtbb、名前が関数テンプレートで指定されたルールに準拠しています。

**メモリセクションの作成**

メモリセクションを作成する方法を示す例については、「[プラグマの挿入によるメモリ内のデータと関数の配置の制御](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ug/control-data-and-function-placement-in-memory-by-inserting-pragmas.html&usg=ALkJrhhTvQ8Ns9ZqL9bt76c_yJR8ZqTw4w)」を参照してください。

**静的および動的に初期化されたデータで使用するストレージクラスを作成する**

この例は、モデル名に依存する名前のファイルにグローバル変数の定義と宣言を配置するストレージクラスを作成する方法を示しています。ストレージクラスの2つのコピーを作成して、1つのコピーをパラメーターデータ（データカテゴリ**モデルパラメーター**）で使用し、1つのコピーを他のデータで使用できるようにします。

通常、生成されたコードは、関数の外部でパラメーターデータを静的に初期化し、モデル初期化関数で他のデータを動的に初期化します。カスタムストレージクラスデザイナーまたは埋め込みコーダーディクショナリを使用してストレージクラスを作成する場合は、**データ初期化**プロパティを設定し て初期化メカニズムを指定します。

Embedded Coderディクショナリでは、ストレージクラスごとにStaticまたはを選択する必要があります Dynamic。パラメータデータ用のストレージクラスのコピーを1つ作成し（Static）、他のデータ用にコピーを1つ作成することを検討してください（Dynamic）。

**ストレージクラスの作成**

1. モデル例を開きrtwdemo\_rollます。
2. [ **Cコード** ]タブが表示されていない場合は、Embedded Coderアプリを開いて[ **Cコード** ]タブをクリックします。
3. モデルダイアグラムの下のコードマッピングエディターで、[埋め込みCoder辞書]アイコンをクリックします。
4. Embedded Coderディクショナリで、[ **追加** ] **を**クリックします 。
5. 新しいストレージクラスに対して、次のプロパティを設定します。
   * **名前**を SigsStates
   * **ヘッダーファイル**に $R\_my\_data.h
   * **定義ファイル**に $R\_my\_data.c

デフォルトでは、**Data Initialization**プロパティはに設定されていますDynamic。これは、ストレージクラスが信号、状態、およびデータストアでの使用に適していることを意味します。

変更を行った後、疑似コードのプレビューが期待どおりに反映されていることを確認します。

1. [ **複製]を**クリックします。新しいストレージクラス SigsStates\_copyが表示されます。
2. 新しいストレージクラスに対して、次のプロパティを設定します。
   * **名前**を Params
   * **データの初期化**へ Static

変更を行った後、疑似コードのプレビューが期待どおりに反映されていることを確認します。

**ストレージクラスの適用とコードの生成**

1. コードマッピングエディターに戻ります。
2. 上の**データのデフォルトの**タブのための**モデルパラメータの**行に**ストレージクラスの** 列を選択Params。
3. 以下のために**内部データ**行、設定**ストレージクラスを**しますSigsStates。
4. 最適化によって生成されたコードから削除されないように、モデルの一部のパラメーターデータ要素を構成します。モデルデータエディターを開きます。
5. [ **パラメータ** ]タブを選択します。
6. モデルで、BasicRollMode サブシステムに移動します。
7. ブロック図を更新します。これで、データテーブルには、モデルで使用されるワークスペース変数に対応する行が含まれます。
8. [ **コンテンツ**の**フィルター]**ボックスの横にある[ **選択**ボタン**を使用し**て**フィルター]を**アクティブにし ます。
9. モデルで、3つのGainブロックを選択します。
10. 図を更新します。
11. モデルデータエディタでは、データテーブルに、変数に対応する3つの行を選択しdispGain、 intGainおよびrateGainモデルワークスペースです。
12. 変数ごとに、[ **ストレージクラス** ]列でを選択しますConvert to parameter object。

モデルデータエディターは、ワークスペース変数をSimulink.Parameterオブジェクトに変換し ます。新しいオブジェクトはストレージクラスを使用します。つまり、オブジェクトは、コードマッピングエディターの**モデルパラメーター**にModel default指定したデフォルトのストレージクラスを取得します。

1. [コンフィギュレーションパラメーター]ダイアログボックスの[ **コード生成]** > [ **コード配置]**ペインで、**[ファイルのパッケージ形式]**をに 設定し**ます**Modular。
2. コードを生成します。
3. コードビューで、ファイルrtwdemo\_roll\_my\_data.cとを 開いて検査します rtwdemo\_roll\_my\_data.h。これらのファイルは、パラメーターオブジェクトおよびサブシステム 内のIntegratorブロックの状態などの一部のブロック状態に対応するグローバル変数を定義および宣言しBasicRollModeます。
4. / \*ストレージクラス 'SigsStates' \* /
5. real32\_T rtFixPtUnitDelay1\_DSTATE;
6. real32\_T rtIntegrator\_DSTATE;
7. int8\_T rtIntegrator\_PrevResetState;
8. / \*ストレージクラス 'Params' \* /
9. real32\_T dispGain = 0.75F;
10. real32\_T intGain = 0.5F;

real32\_T rateGain = 2.0F;

**パッケージ内のコード生成定義を参照してください**

Embedded Coderディクショナリを構成して、パッケージに保存したコード生成定義を参照できます（「[コード定義を作成してデフォルト設定を上書きする」を](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ug/define-storage-class-memory-section-and-function-template-settings-for-a-software-architecture.html&usg=ALkJrhjZ24TZTUcg062CLOaujh22tfnz1w" \l "mw_d717c46d-9c55-4d08-a43d-27bd007453ad)参照）。これらの定義は、コードマッピングエディターで選択できるようになります。この例では、組み込みのrtwdemo\_rollサンプルパッケージに格納されている定義を参照するようにのEmbedded Coder辞書を設定し ますECoderDemos。

1. のEmbedded Coder辞書を開きrtwdemo\_rollます。手順については、[ストレージクラスの作成と確認を](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ref/embeddedcoderdictionary.html&usg=ALkJrhgBvIGIuFTu-XrpdBjX_V74z3oJcA" \l "mw_835f0ffc-7ddd-4787-b9a2-a6b53299f15d)ご覧ください。
2. [Embedded Coder Dictionary]ウィンドウで、[ **パッケージの管理** ]をクリックします。
3. [パッケージの管理]ダイアログボックスで、[ **更新** ]をクリックします。**[パッケージ**の**選択]** ドロップダウンリストにさらにオプションが表示されるまで待ちます。
4. **[Select package** to ECoderDemos] を設定して 、[ **Load** ]をクリックします 。

[埋め込みコーダーディクショナリ]ウィンドウの[ **ストレージクラス** ]タブの表に、ECoderDemosパッケージで定義されているストレージクラスが表示されます。これで、で rtwdemo\_roll、 [ **データのデフォルト** ]タブのコードマッピングエディターでこれらのストレージクラスを選択できます。

1. パッケージをアンロードするには、[パッケージの管理]ダイアログボックスの**[パッケージ**の**選択** ]ドロップダウンリストで**パッケージを選択**し、[ **アンロード** ]をクリックします 。

**Simulinkデータディクショナリを使用してモデル間でコード生成定義を共有する**

データディクショナリを使用してモデル間でコード生成定義を共有する方法を示す例については、「[モデル間でEmbedded Coderディクショナリ定義](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ug/deploy-code-generation-definitions.html&usg=ALkJrhiSeAPtmrvQ4LV5wqNbv1UOJSAbDQ#mw_5218a566-c71a-4f64-8f13-d0126494c6fc)を[共有する](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ug/deploy-code-generation-definitions.html&usg=ALkJrhiSeAPtmrvQ4LV5wqNbv1UOJSAbDQ#mw_5218a566-c71a-4f64-8f13-d0126494c6fc)」を参照してください。

**共有コーダーディクショナリでのデフォルトコードマッピングの構成**

共有Embedded Coder辞書で既定のコードマッピングを構成する方法を示す例については、「[共有辞書で既定のコードマッピング](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ug/deploy-code-generation-definitions.html&usg=ALkJrhiSeAPtmrvQ4LV5wqNbv1UOJSAbDQ" \l "mw_5cbdccb9-584f-46f3-b50f-b91934e30032)を[構成する](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ug/deploy-code-generation-definitions.html&usg=ALkJrhiSeAPtmrvQ4LV5wqNbv1UOJSAbDQ#mw_5cbdccb9-584f-46f3-b50f-b91934e30032)」を参照してください。

**関連する例**

* [ソフトウェアアーキテクチャのストレージクラス、メモリセクション、および関数テンプレートを定義する](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ug/define-storage-class-memory-section-and-function-template-settings-for-a-software-architecture.html&usg=ALkJrhjZ24TZTUcg062CLOaujh22tfnz1w)
* [コード生成定義をデプロイする](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ug/deploy-code-generation-definitions.html&usg=ALkJrhiSeAPtmrvQ4LV5wqNbv1UOJSAbDQ)
* [モデル間でデフォルトのコード生成設定を共有およびコピーすることにより、ソフトウェアアーキテクチャに準拠](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ug/conform-to-a-software-architecture-by-sharing-and-copying-default-code-generation-settings-between-models.html&usg=ALkJrhiUZwUtc1Sc_1vzKviFmaEGaD5yig)
* [さまざまなモデル階層コンテキスト用の柔軟なストレージクラス](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ug/define-a-storage-class-with-different-settings-based-on-model-hierarchy-context.html&usg=ALkJrhgfZhOHkPU54yg5rkXAxt_FhsMppA)

**パラメーター**

これらのプロパティは、Embedded Coderディクショナリウィンドウのプロパティインスペクターペインに表示されます。表では、いくつかのプロパティが列として表示され、バッチ編集を容易にします。

**ストレージクラス**

**Name—ストレージクラスの名前  
StorageClass1（デフォルト）| テキスト**

**Description—ストレージクラステキストの目的と機能**

**Source —ストレージクラス定義テキストの場所**

**Data Access—データにアクセスするための仕様  
Direct（デフォルト）|Function**

**Data Scope—データ定義を生成するための仕様  
Exported（デフォルト）|Imported**

**Header File—データを宣言するヘッダーファイルの名前  
$N.h（デフォルト）| テキスト**

**Definition File—データを定義するソースファイルの名前  
$N.c（デフォルト）| テキスト**

**Access Mode—関数を介してデータにアクセスするための仕様  
Value（デフォルト）|Pointer**

**Allowed Access—関数を介したデータへのアクセスを許可する仕様  
Read/Write（デフォルト）| Read Only|Write Only**

**Name of Getter— get関連データをフェッチする関数の名前  
get\_$N$M（デフォルト）| テキスト**

**Name of Setter— set関連データを変更する関数の名前  
set\_$N$M（デフォルト）| テキスト**

**Use different property settings for single-instance and multi-instance data—個別のストレージ設定を割り当てるための仕様  
off（デフォルト）|on**

**Storage Type—データを構造体に集約するための仕様  
Unstructured（デフォルト）|Structured**

**Type Name—構造タイプの名前  
$R$N$G$M（デフォルト）| テキスト**

**Instance Name—構造体変数の名前  
$N$G$M（デフォルト）| テキスト**

**Data Initialization—データを初期化する方法  
Dynamic（デフォルト）| Static| None**

**Memory Section—データを割り当てるメモリ内の場所  
None（デフォルト）| 既存のメモリセクション**

**Preserve array dimensions—多次元配列の次元を維持するための仕様  
off（デフォルト）|on**

**Const— const修飾子を適用するための仕様  
off（デフォルト）|on**

**Volatile— volatile修飾子を適用するための仕様  
off（デフォルト）|on**

**Other Qualifier—カスタム修飾子テキストを適用するための仕様**

**Parameters—モデルパラメータでの使用を許可するかどうか  
off（デフォルト）|on**

**Signals—モデル信号での使用を許可するかどうか  
on（デフォルト）|off**

**関数カスタマイズテンプレート**

**Name—関数テンプレートの名前  
FunctionTemplate1（デフォルト）| テキスト**

**Description—関数テンプレートテキストの目的と機能**

**Source —関数テンプレート定義テキストの場所**

**Function Name—生成された関数の名前  
$R$N（デフォルト）| テキスト**

**Memory Section—関数を割り当てるメモリ内の場所  
None（デフォルト）| 既存のメモリセクション**

**メモリセクション**

**Name—メモリセクションテキストの名前**

**Description—メモリセクションテキストの目的と機能**

**Source —メモリセクション定義テキストの場所**

**Comment—生成されたコードテキストに挿入するコメント**

**Pre Statement—データまたは関数コードテキストの前に挿入するコード**

**Post Statement—データまたは関数コードテキストの後に挿入するコード**

**Statements Surround—データと関数を個別にまたはグループでラップする仕様  
Each variable（デフォルト）|Group of variables**

**制限事項**

* Embedded Coderディクショナリで作成したストレージクラスまたは関数のカスタマイズテンプレートは、パッケージからロードするメモリセクションを使用できません（[「パッケージ内のコード生成定義」を参照](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ref/embeddedcoderdictionary.html&usg=ALkJrhgBvIGIuFTu-XrpdBjX_V74z3oJcA#mw_a95f157e-7abb-48e1-8e18-e8bcbd9241f1)）。Embedded Coderディクショナリで定義されたメモリセクションを使用します。
* .mdl モデルファイルにコード生成定義を作成することはできません。
* データディクショナリ（.slddファイル）のEmbedded Coderディクショナリでのコード生成定義の追加の制限については、「[コード生成定義の展開」を](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ug/deploy-code-generation-definitions.html&usg=ALkJrhiSeAPtmrvQ4LV5wqNbv1UOJSAbDQ)参照してください 。

**こちらもご覧ください**

[コードマッピングエディター](https://translate.googleusercontent.com/translate_c?depth=1&hl=ja&pto=aue&rurl=translate.google.co.jp&sl=en&sp=nmt4&tl=ja&u=https://de.mathworks.com/help/releases/R2020a/ecoder/ref/codemappingseditor.html&usg=ALkJrhiy2gG9isGNGy4yhBTJ4xdjnYkqSA)

**R2018aで導入**