

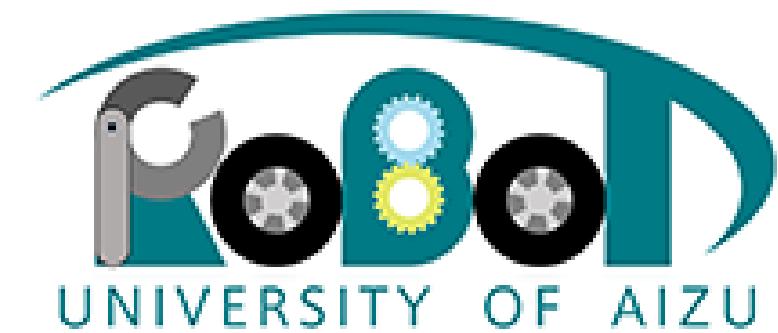
9コマ目

RTCBUILDERを用いた

コンポーネント作成

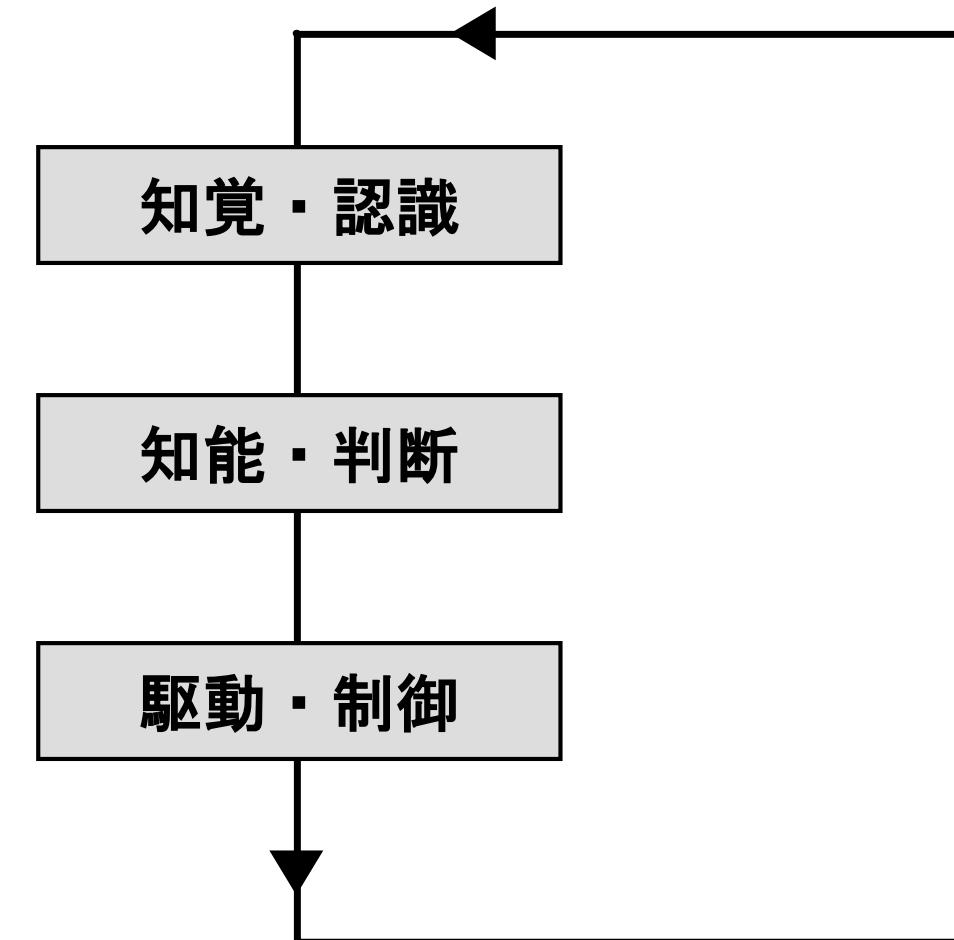


ロボットプログラミングの考え方



ロボットプログラミング

- ロボットの3要素
 1. 知覚・認識
 - カメラ, 速度センサなど
 2. 知能・判断
 - プランニングなど
 3. 駆動・制御
 - アクチュエータ（モータ）など



プログラムの種類

- フロー駆動型
 - プログラム記述通りに、上から下にプログラムを実行する
- イベント駆動型プログラム
 - イベントに対して実行される内容が変化する

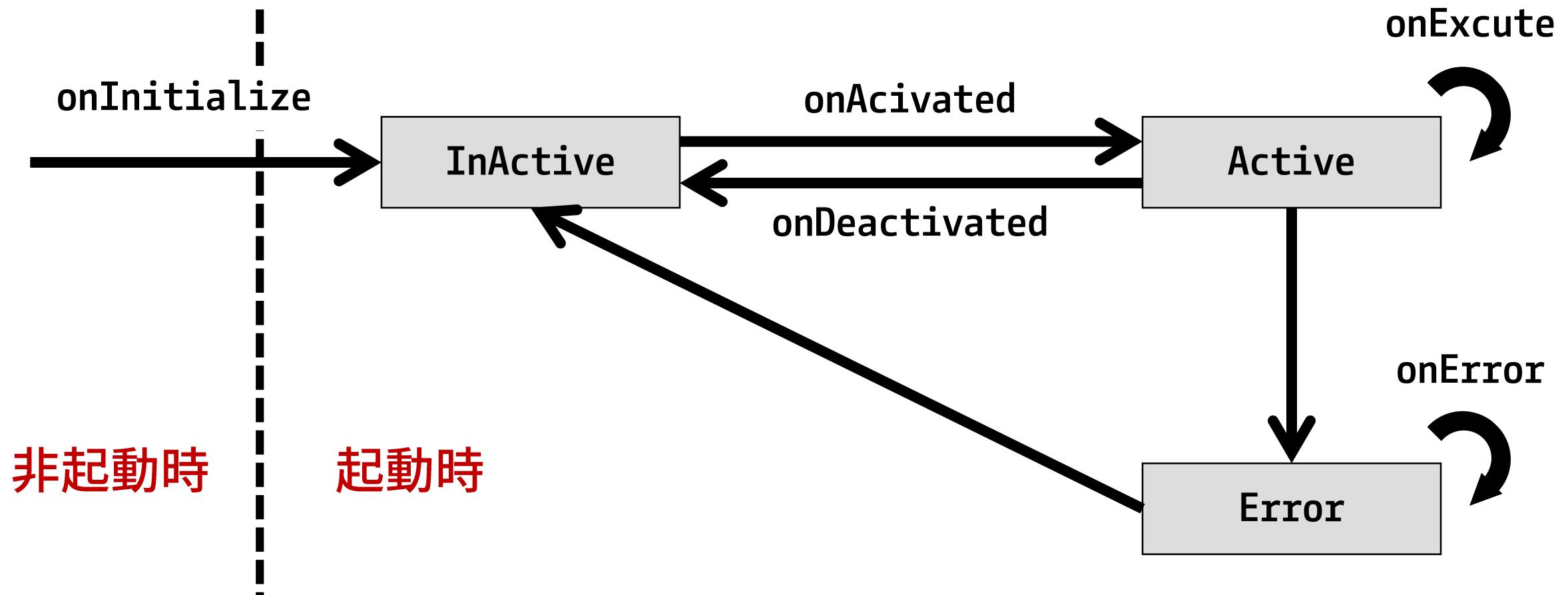
OpenRTM-aistの場合

- ・コールバック関数によるイベント駆動型プログラム
 - ・以下の決まった関数を使用してプログラムを構成する

関数名	処理内容
onInitialize	コンポーネント起動時に実行し、初期化の処理をする
onAcivated	コンポーネント実行時に1度だけ実行する
onExcute	コンポーネント実行時に周期的に実行する
onDeactivated	コンポーネント停止時に1度だけ実行する
onError	コンポーネントがERROR状態のときに周期的に実行する

OpenRTM-aistの場合

- ・コールバック関数の起動タイミング



安全確認型と危険検出型について

- インターロック：条件が満たされなければ動作しない仕組み
 - 安全確認型：安全が確認されているときのみ動作する
 - 安全が確認されなければ動作しない
 - 危険検出型：危険が検知されれば動作を停止する
 - 危険が検知されなければ動作を停止しない
- ロボットがどのように動作するかは、プログラマの設計次第

安全確認型と危険検出型の比較

例：信号を渡るとき

安全確認型（青信号で渡る）：
青以外の信号では『止まる』

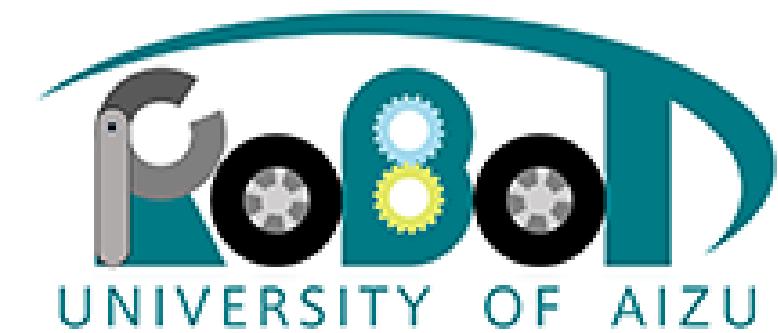
- 青信号：渡る
- 黄信号：止まる
- 赤信号：止まる
- 故障時：止まる

危険確認型（赤信号で止まる）：
赤以外の信号では『渡る』

- 青信号：渡る
- 黄信号：渡る
- 赤信号：止まる
- 故障時：渡る

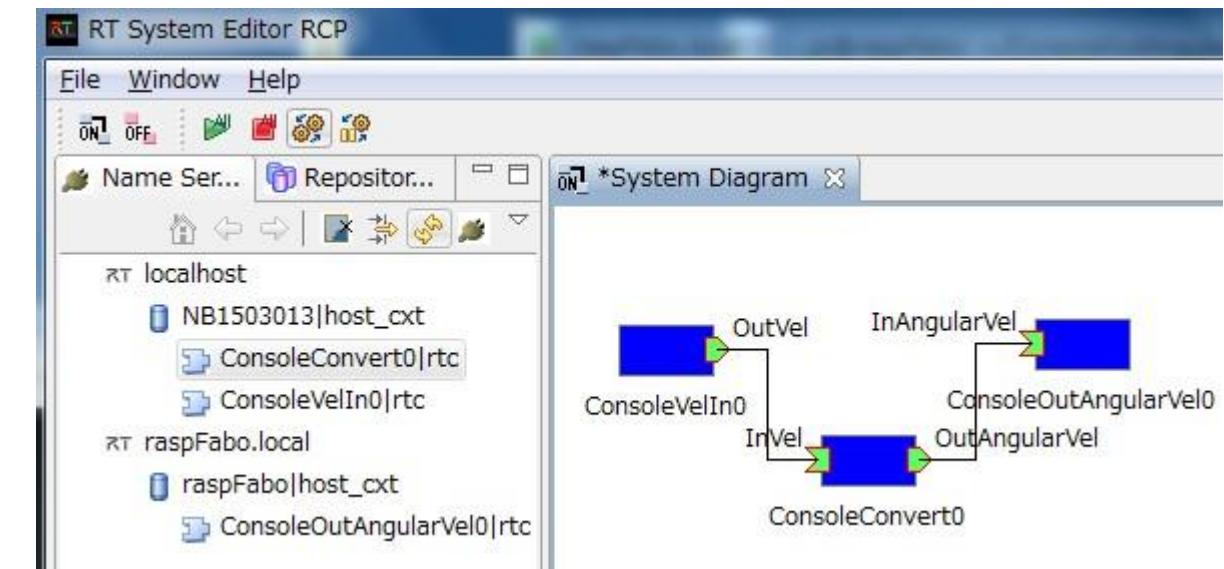
故障が起きた際に、
どのようなことが起こるかを見据えた制御を考えないと
危険に正しく対処することはできない

RTコンポーネント作成



作成するコンポーネント

- 前方速度と角速度を入力し、各タイヤの角速度を表示するシステムを作成
- 以下の3コンポーネントを作成
 - ConsoleVelIn**
 - 前方速度と角速度を入力
 - ConsoleConvert**
 - 各タイヤの角速度を計算
 - ConsoleOutAngularVel**
 - 各タイヤの角度度を表示

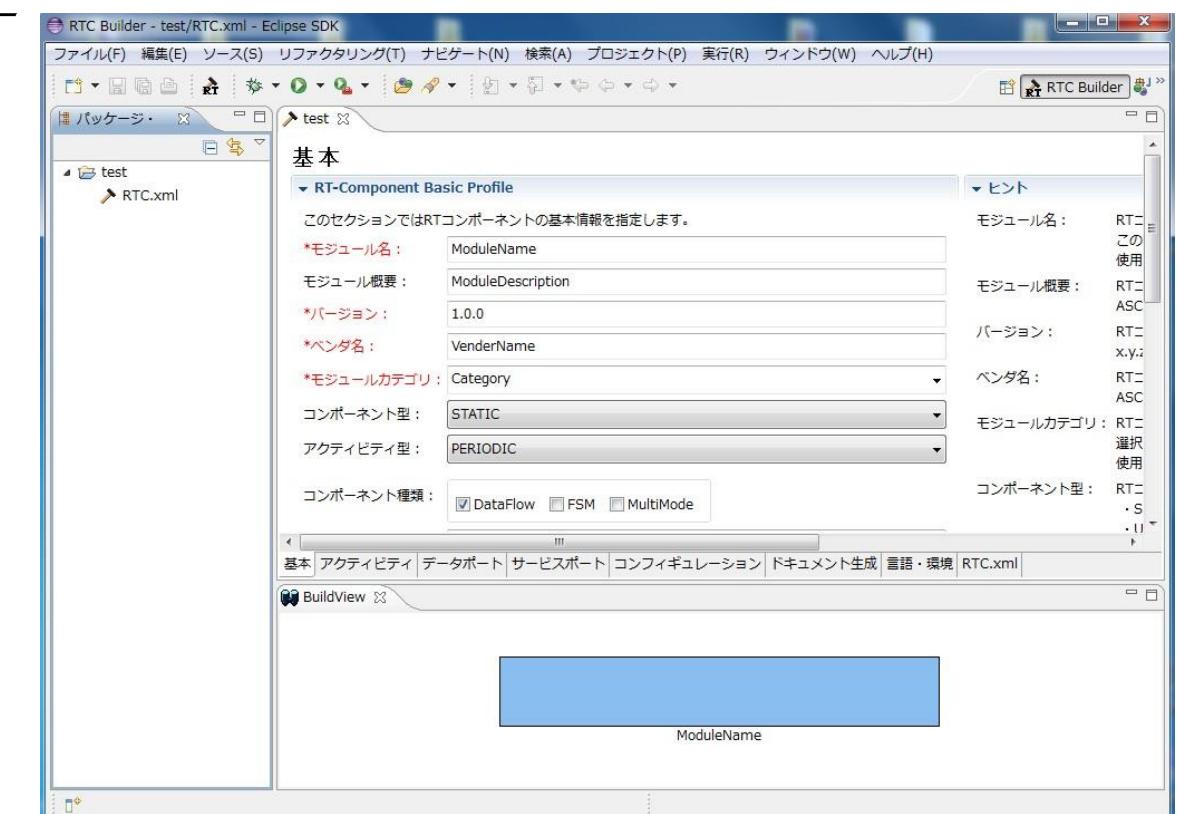


作成手順

- RTCBuilder
 - ひな型作成
 - 基本
 - データポート
 - コンフィギレーション
 - 言語・環境
- プログラムの編集
 - Python
 - ソースコード編集

RTCBuilder概要

- ・コンポーネントの情報を入力し雛形を生成するためのツール
 - ・コンポーネントの仕様部分を設定
 - ・各言語向けRTCの雛型を作成
 - ・C++
 - ・Java
 - ・Python

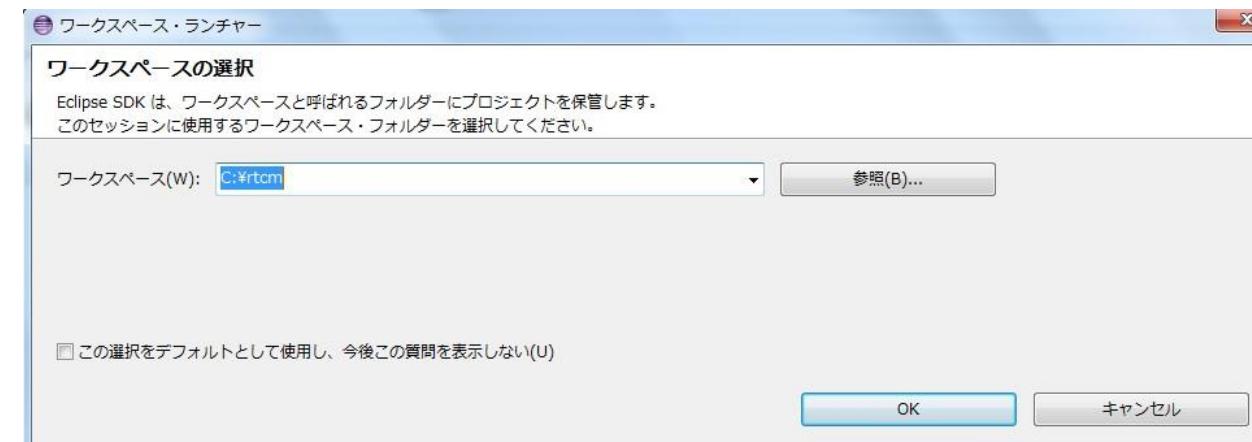


ConsoleVellInコンポーネントの作成



RTCBUILDER起動

- Windowsの場合
 - スタートメニューから「OpenRTM-aist 1.1.2」→「tools」の[OpenRTP]から起動する
 - 起動後はワークスペースの選択を行う
 - 今回は[C:\ rtcws]と記入
※パスは半角英数字（全角文字・日本語が入らないよう）にする



初回起動時のみの操作

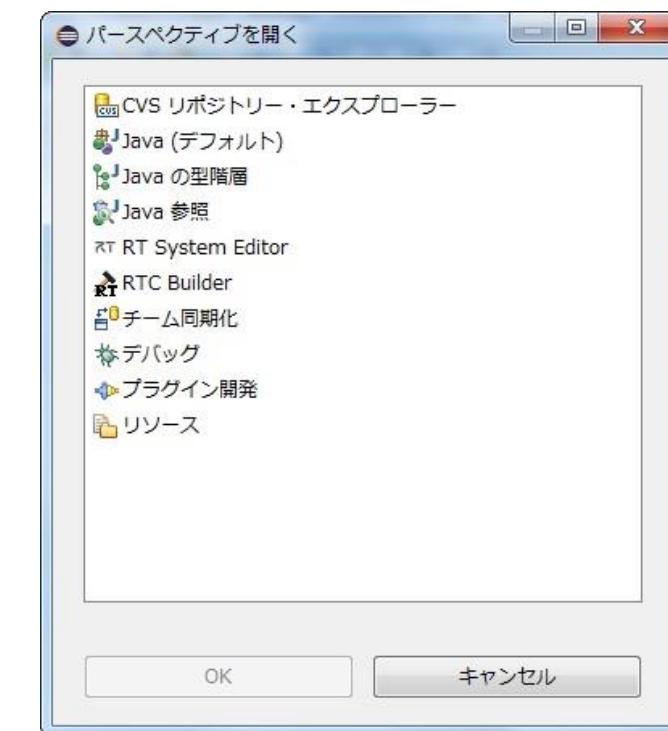
1. 初回起動時に出現するようこそタブの×ボタンをクリックし、その画面を消す



2. [パースペクティブを開く]を選択する

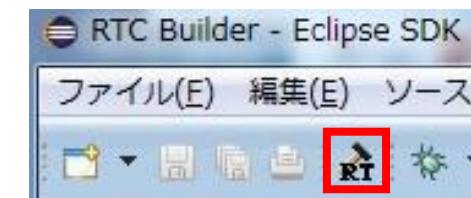


3. 一覧から[RTCBuilder]を選択する

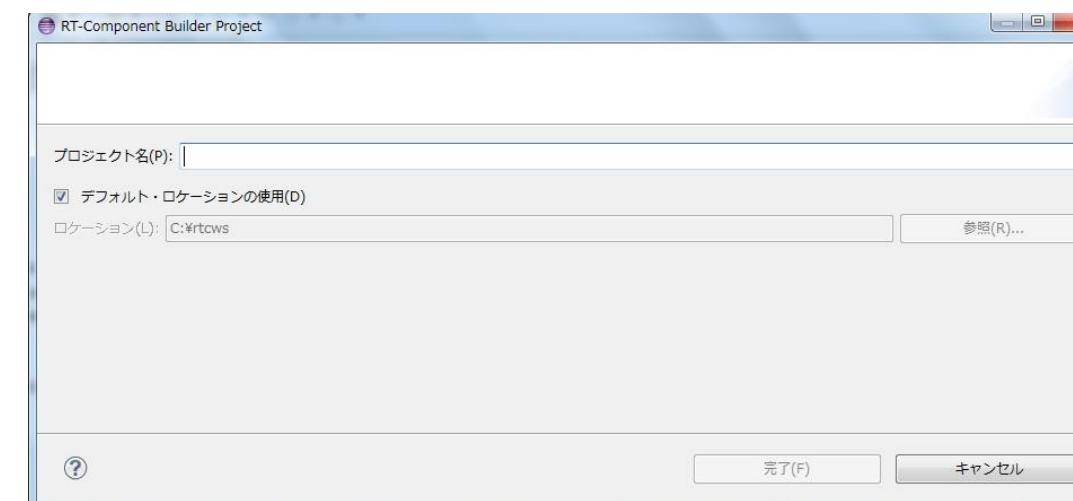


プロジェクトの作成

- ツールバー内の[RTとカナヅチ]のアイコンをクリック



- プロジェクト名を記入 (**ConsoleVelIn**)



基本プロファイル

- コンポーネント基本情報の設定

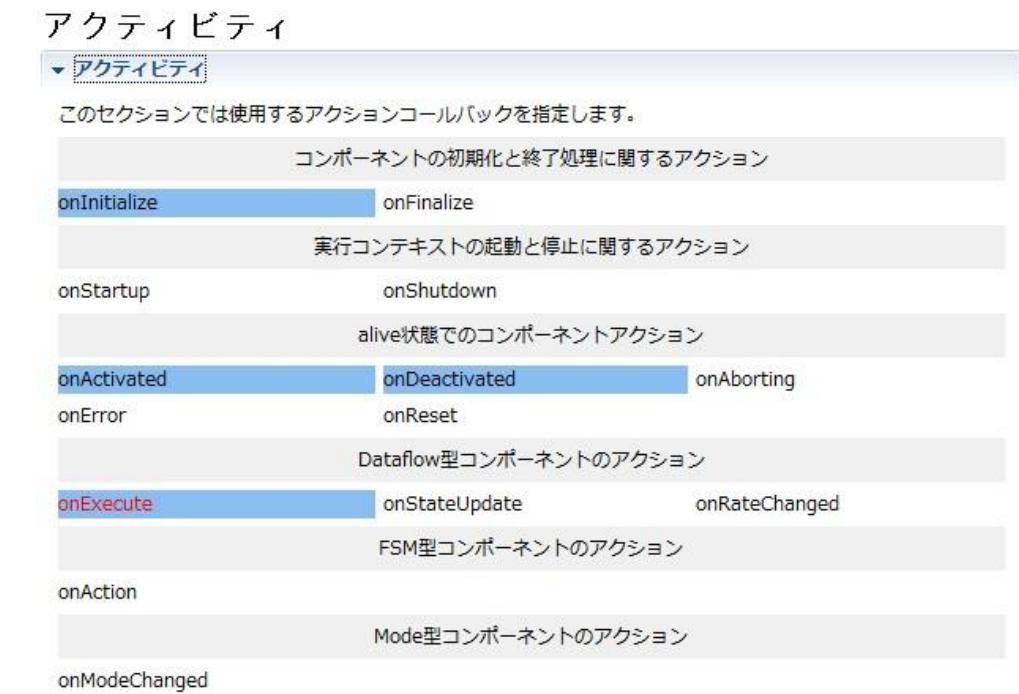
モジュール名	ConsoleVelIn	基本
モジュール概要	任意(ConsoleVelIn component)	RT-Component Basic Profile このセクションではRTコンポーネントの基本情報を指定します。
バージョン	1.0.0	*モジュール名 : ConsoleVelIn
ベンダ名	任意	モジュール概要 : ConsoleVelIn component
モジュールカテゴリ	任意(Category)	*バージョン : 1.0.0
コンポーネント型	STATIC	*ベンダ名 : VenderName
アクティビティ型	PERIODIC	*モジュールカテゴリ : Category
コンポーネントの種類	DataFlow	コンポーネント型 : STATIC
最大インスタンス数	1	アクティビティ型 : PERIODIC
実行型	PeriodicExecutionContext	コンポーネント種類 : <input checked="" type="checkbox"/> DataFlow <input type="checkbox"/> FSM <input type="checkbox"/> MultiMode
実行周期	1000.0	最大インスタンス数 : 1
		実行型 : PeriodicExecutionContext
		実行周期 : 1000.0

アクティビティ・プロファイル

- コンポーネントで使用予定のアクションコールバックを設定
 - 使用するアクションをダブルクリック（青背景にすることで選択する）
 - ※クリック1回目で文字が赤くなり、クリック2回目で背景が青くなる

選択アクションコールバック

- onInitialize
- onActivated
- onExecute
- onDeactivated



データポート・プロファイル

- RTコンポーネントのデータポートを設定

- 追加したいポートの[Add]ボタンを押し、
ポートを追加し、直接名称を変更

*ポート名 (OutPort)	Add
OutVel	Delete

- ポートのデータ型と変数名を入力

*データ型	RTC::TimedVelocity2D
変数名	OutValue
表示位置	RIGHT

データポート

DataPortプロファイル

このセクションではRTコンポーネントのDataPort(データポート)の情報を設定します。

*ポート名 (InPort)	Add	*ポート名 (OutPort)	Add
	Delete		Delete

Detail

このセクションではデータポート毎の概要を説明するドキュメントを記述します。
上のデータポートを選択すると、それとのドキュメントが記述できます。

ポート名 :

*データ型 : RTC::Acceleration2D
変数名 :
表示位置 : LEFT

Documentation

概要説明 :

データ型 :
データ数 :

意味 :

単位 :

発生頻度、周期 :

処理頻度、周期 :

基本 アクティビティ データポート サービスポート コンフィギュレーション ドキュメント生成 言語・環境 RTC.xml

OutPort

- ポート名 : OutVel
- データ型 : RTC::TimedVelocity2D
- 変数名 : OutValue
- 表示位置 : RIGHT

サービスポート・プロファイル

- RTコンポーネントのサービスポートを設定
 - このコンポーネントでは使用しないので、スキップする

サービスポート



The screenshot shows the configuration interface for an RT-Component Service Port Profile. On the left, there is a list titled "RT-Component Service Ports" containing a single entry: "sv_name". On the right, there is a detailed configuration panel titled "RT-Component Service Port Profile". It includes fields for "Port Name" (set to "sv_name") and "Display Position" (set to "LEFT"). Below these are sections for "Documentation", "Summary Description", and "I/F Summary Description", each with a large text input area.

RT-Component Service Ports

sv_name

Add Port
Add Interface
Delete

RT-Component Service Port Profile

このセクションではRTコンポーネントのServicePortの情報を設定します。

*ポート名 : sv_name

表示位置 : LEFT

▼ Documentation

概要説明:

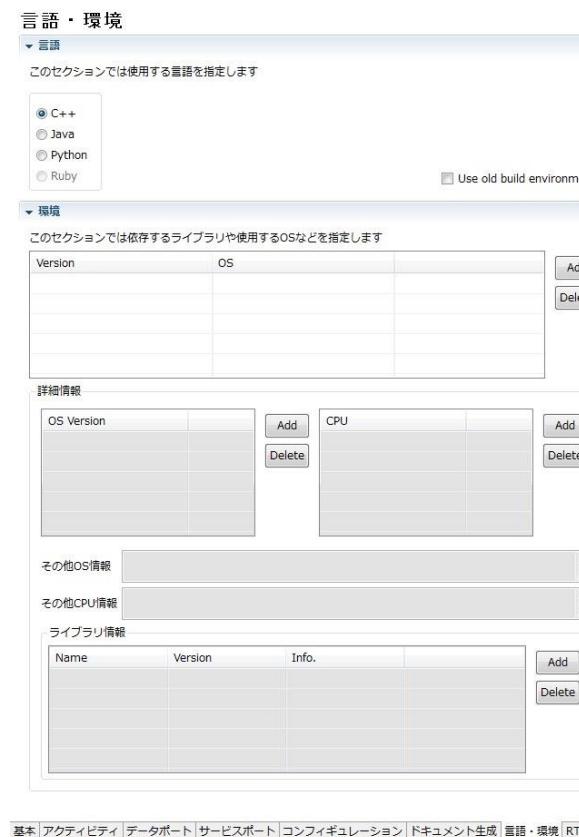
I/F概要説明 :

コンフィギュレーション・プロファイル

- RTコンポーネントのコンフィギュレーションを設定
 - このコンポーネントでは使用しないので、スキップする

言語・環境・プロファイル

- RTコンポーネントの実装する言語、動作環境に関する情報を設定



Pythonを選択する

コード生成

• 設定内容を確認

基本

- モジュール名 : ConsoleVelIn
- モジュール概要 : 任意(ConsoleVelIn component)
- バージョン : 1.0.0
- ベンダ名 : 任意
- モジュールカテゴリ : 任意(Category)
- コンポーネント型 : STATIC
- アクティビティ型 : PERIODIC
- コンポーネントの種類 : DataFlow
- 最大インスタンス数 : 1
- 実行型 : PeriodicExecutionContext
- 実行周期 : 1000.0

選択アクションコールバック

- onInitialize
- onActivated
- onExecute
- onDeactivated

言語・環境

- Python

OutPort

- ポート名 : OutVel
- データ型 : RTC::TimedVelocity2D
- 変数名 : OutValue
- 表示位置 : RIGHT

▼ コード生成とパッケージ化

コードの生成およびパッケージ化を行います。

- すべての設定を終えたらコード生成
 - コード生成のボタンをクリック

プログラムの編集



コピー用テキスト

- 以下URLからコピー用のテキストが配置されている
<https://rtc-fukushima.jp/wp/wp-content/uploads/2018/11/09Program.zip>
- 以下のファイルがあることを確認してください
 - ConsoleConvert.txt
 - ConsoleOutAngularVel.txt
 - ConsoleVelIn.txt
- このテキストからコピー＆ペーストがしづらい方はzipファイル内のテキストからコピー＆ペーストしてください

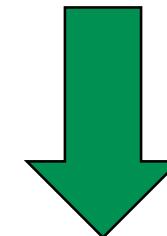
Python

- Pythonプログラムの編集手順
 - ConsoleVelIn.pyの各関数を編集
 - `_init_`
 - 起動時によばれ、ポートの初期化を行う
 - `onActivated`
 - 非アクティブ状態からアクティブ化されるとき1度だけよばれる
 - `onDeactivated`
 - アクティブ状態から非アクティブ化されるとき1度だけよばれる
 - `onExecute`
 - アクティブ状態時に周期的によばれる
 - Pythonでは宣言した関数以外でも変数を使用したい場合、変数の先頭に **self.** を付ける

__init__関数の変更

- __init__ 関数内のInPort, OutPortの初期化を
以下のように変更する

```
self._d_OutValue = RTC.TimedVelocity2D(*OutValue_arg)
```



```
self._d_OutValue = RTC.TimedVelocity2D(RTC.Time(0,0), RTC.Velocity2D(0.0, 0.0, 0.0))
```

ひな形の宣言と変数の関係

- RTCBUILDERで宣言した変数とポート名はPythonプログラムでは以下の名前で使用できる

InPortポート名	self._ポート名In
InPort変数名	self._d_変数名
OutPortポート名	self._ポート名Out
OutPort変数名	self._d_変数名

- このコンポーネントでは以下のようになる

OutPortポート名 : OutVel	self._OutVelOut
OutPort変数名 : OutValue	self._d_OutValue

RTC.TimedVelocity2D型について

- RTC.TimedVelocity2D型は構造体で以下のように定義されている

データ型	変数名	意味
RTC.Velocity2D	data	速度情報
RTC.Time	tm	タイムスタンプ

- RTC.Velocity2D型は構造体で以下のように定義されている

データ型	変数名	意味
Double	vx	並進速度(前方) [m/s]
Double	vy	並進速度(横方) [m/s]
Double	va	角速度 [rad/s]

RTC.TimedVelocity2D型について

- RTC.Time型は構造体で以下のように定義されている

データ型	変数名	意味
Int	sec	秒
Int	nsec	ナノ秒

- 使用例

```
self._d_OutValue.data.vx = 0.0 # vxに0.0を入れる
```

onActivated関数全文

- onActivated関数を以下のように編集する

```
def onActivated(self, ec_id):  
    print "onActivated"  
    self._d_OutValue.data.vx = 0.0 # vxを初期化  
    self._d_OutValue.data.va = 0.0 # vaを初期化  
    self._d_OutValue.data.vy = 0.0 # vyを初期化  
    return RTC.RTC_OK
```

onDeactivated関数全文

- onDeactivated関数を以下のように編集する

```
def onDeactivated(self, ec_id):  
    print "onDeactivated"  
    return RTC.RTC_OK
```

onExecute関数 | 並進速度（前方）の入力

- 並進速度（前方）の値をコンソールから入力するプログラム

```
print "input Vx:"  
Vx = raw_input() # コンソールからvxの値を入力  
print "Vx:"+ Vx
```

onExecute関数 | 角速度の入力

- ・角速度の値をコンソールから入力するプログラム

```
print "input Va:"  
Va = raw_input() # コンソールからvaの値を入力  
print "Va:"+ Va
```

onExecute関数 | OutPortから値の出力

- コンポーネントから値を出力する

```
# vxの値をfloatにcast後に変数に入力  
self._d_OutValue.data.vx = float(Vx)  
# vaの値をfloatにcast後に変数に入力  
self._d_OutValue.data.va = float(Va)  
self._d_OutValue.data.vy = 0.0  
self._OutVelOut.write() # OutPortから値を出力
```

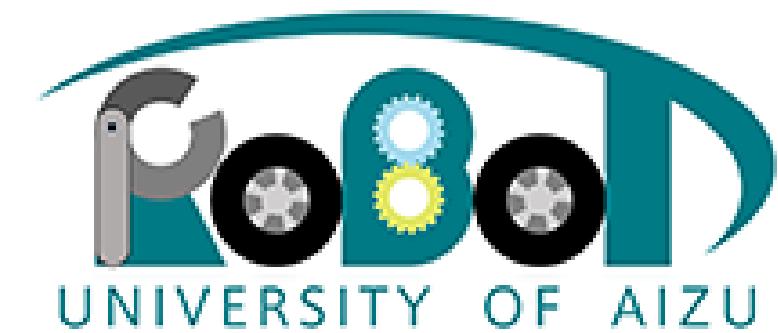
- コンポーネントから値を出力するにはOutPortの変数に値を入力してwrite関数で出力します

onExecute関数全文

- onExecute内を以下のように編集する

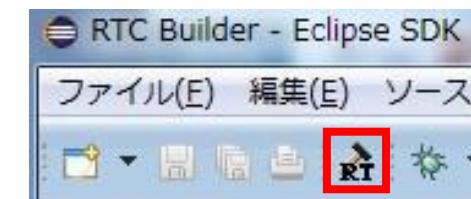
```
def onExecute(self, ec_id):  
    print "input Vx: "  
    Vx = raw_input() # コンソールからvxの値を入力  
    print "Vx:" + Vx  
    print "input Va: "  
    Va = raw_input() # コンソールからvaの値を入力  
    print "Va:" + Va  
    # vxの値をfloatにcast後に変数に入力  
    self._d_OutValue.data.vx = float(Vx)  
    # vaの値をfloatにcast後に変数に入力  
    self._d_OutValue.data.va = float(Va)  
    self._d_OutValue.data.vy = 0.0  
    self._OutVelOut.write() # OutPortから値を出力  
    return RTC.RTC_OK
```

ConsoleConvertコンポーネントの作成

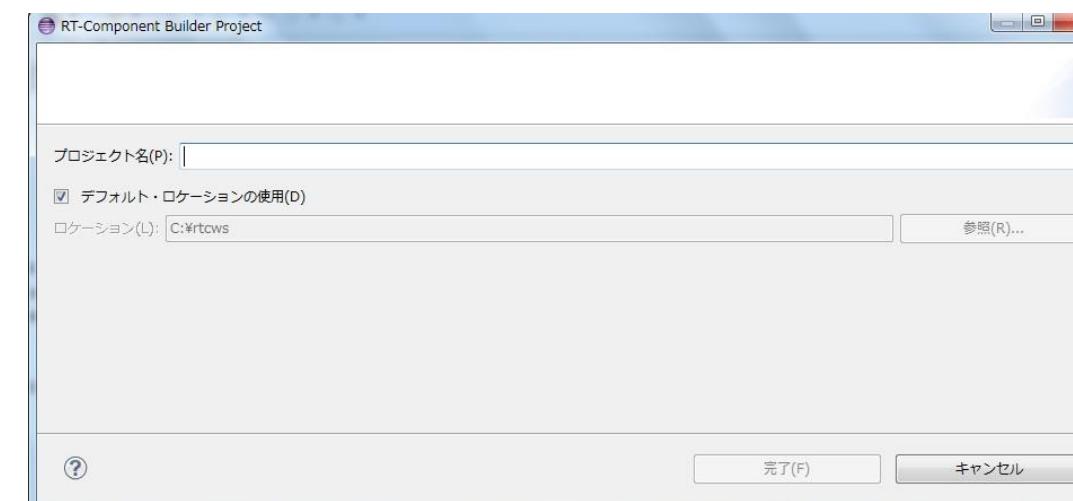


プロジェクトの作成

- ツールバー内の[RTとカナヅチ]のアイコンをクリック



- プロジェクト名を記入 (**ConsoleConvert**)



基本プロファイル

- コンポーネント基本情報の設定

モジュール名	ConsoleConvert
モジュール概要	任意(ConsoleConvert component)
バージョン	1.0.0
ベンダ名	任意
モジュールカテゴリ	任意(Category)
コンポーネント型	STATIC
アクティビティ型	PERIODIC
コンポーネントの種類	DataFlow
最大インスタンス数	1
実行型	PeriodicExecutionContext
実行周期	1000.0

基本

▼ RT-Component Basic Profile

このセクションではRTコンポーネントの基本情報を指定します。

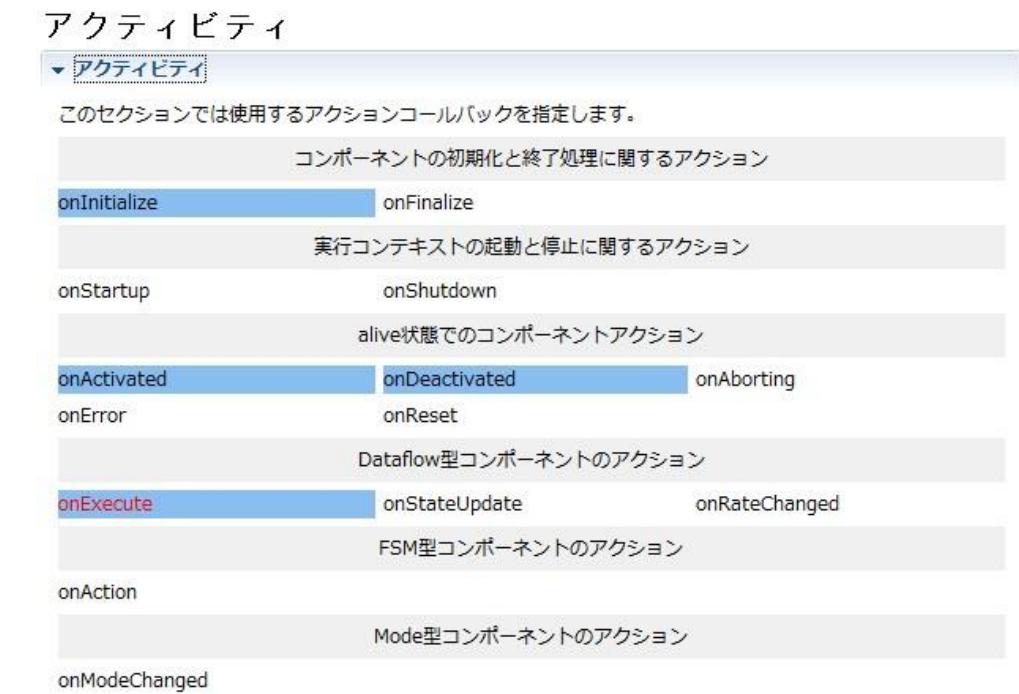
*モジュール名 :	ConsoleConvert
モジュール概要 :	ConsoleConvert component
*バージョン :	1.0.0
*ベンダ名 :	VenderName
*モジュールカテゴリ :	Category
コンポーネント型 :	STATIC
アクティビティ型 :	PERIODIC
コンポーネント種類 :	<input checked="" type="checkbox"/> DataFlow <input type="checkbox"/> FSM <input type="checkbox"/> MultiMode
最大インスタンス数 :	1
実行型 :	PeriodicExecutionContext
実行周期 :	1000.0

アクティビティ・プロファイル

- コンポーネントで使用予定のアクションコールバックを設定
 - 使用するアクションをダブルクリック（青背景にすることで選択する
 - ※クリック1回目で文字が赤くなり、クリック2回目で背景が青くなる

選択アクションコールバック

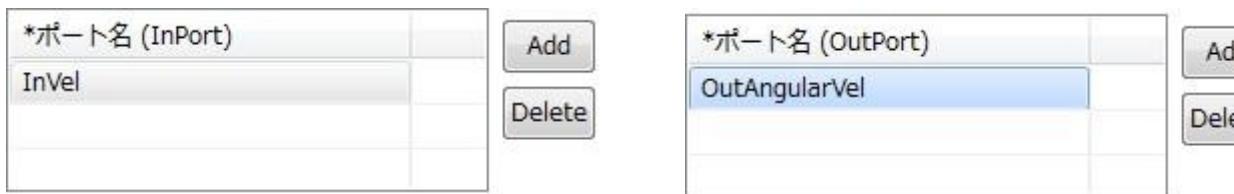
- onInitialize
- onActivated
- onExecute
- onDeactivated



データポート・プロファイル

- RTコンポーネントのデータポートを設定

- 追加したいポートの[Add]ボタンを押し、
ポートを追加し、直接名称を変更



- ポートのデータ型と変数名を入力

*データ型	RTC::TimedVelocity2D
変数名	InValue
表示位置	LEFT
*データ型	RTC::TimedFloatSeq
変数名	OutValue
表示位置	RIGHT

InPort

- ポート名 : InVel
- データ型 : RTC::TimedVelocity2D
- 変数名 : InValue
- 表示位置 : LEFT

OutPort

- ポート名 : OutAngularVel
- データ型 : RTC::TimedFloatSeq
- 変数名 : OutValue
- 表示位置 : RIGHT

サービスポート・プロファイル

- RTコンポーネントのサービスポートを設定
 - このコンポーネントでは使用しないので、スキップする

サービスポート



The screenshot shows the configuration interface for an RT-Component Service Port Profile. On the left, there is a list titled "RT-Component Service Ports" with a single entry "sv_name". On the right, there is a detailed configuration panel titled "RT-Component Service Port Profile". It contains the following fields:

- *ポート名 :
- 表示位置 :
- Documentation section:
 - 概要説明: (Large text area)
 - I/F概要説明 : (Large text area)

Below the documentation sections are two large scrollable text areas for "概要説明" and "I/F概要説明".

コンフィギュレーション・プロファイル

- RTコンポーネントのコンフィギュレーションを設定

- [Add]ボタンを押しコンフィギュレーションを追加し、直接名称を変更します

*名称 Treadwidth	Add
radius	Delete

- コンフィギュレーションの各パラメータを入力

パラメータ名 : Treadwidth	パラメータ名 : radius
*データ型 float	*データ型 float
*デフォルト値 0.046	*デフォルト値 0.0275
変数名 : width	変数名 : radius
単位 : m	単位 : m
制約条件:	制約条件:
Widget: text	Widget: text
Step:	Step:

Treadwidth

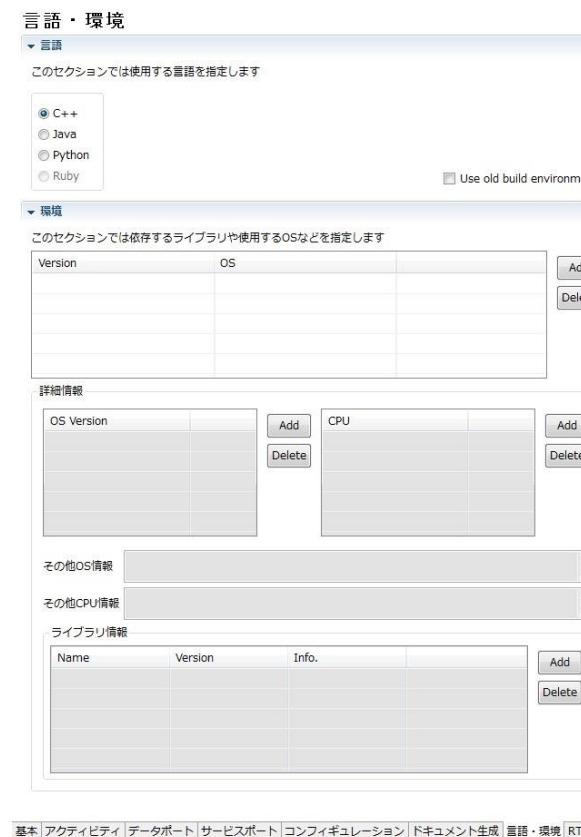
- データ型 : float
- デフォルト値 : 0.046
- 変数名 : width
- 単位 : m

radius

- データ型 : float
- デフォルト値 : 0.0275
- 変数名 : radius
- 単位 : m

言語・環境・プロファイル

- RTコンポーネントの実装する言語、動作環境に関する情報を設定



Pythonを選択

コード生成

• 設定内容を確認

基本

- モジュール名 : ConsoleConvert
- モジュール概要 : 任意(ConsoleConvert component)
- バージョン : 1.0.0
- ベンダ名 : 任意
- モジュールカテゴリ : 任意(Category)
- コンポーネント型 : STATIC
- アクティビティ型 : PERIODIC
- コンポーネントの種類 : DataFlow
- 最大インスタンス数 : 1
- 実行型 : PeriodicExecutionContext
- 実行周期 : 1000.0

選択アクションコールバック

- onInitialize
- onActivated
- onExecute
- onDeactivated

InPort

- ポート名 : InVel
- データ型 : RTC::TimedVelocity2D
- 変数名 : InValue
- 表示位置 : LEFT

OutPort

- ポート名 : OutAngularVel
- データ型 : RTC::TimedFloatSeq
- 変数名 : OutValue
- 表示位置 : RIGHT

コード生成

- 設定内容を確認

Treadwidth

- データ型 : float
- デフォルト値 : 0.046
- 変数名 : width
- 単位 : m

radius

- データ型 : float
- デフォルト値 : 0.0275
- 変数名 : radius
- 単位 : m

言語・環境

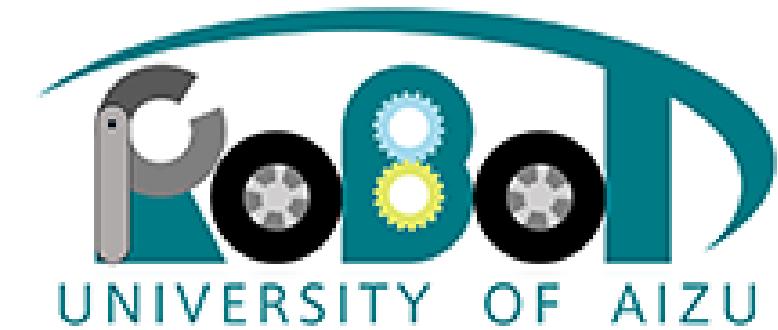
- Python

- すべての設定を終えたらコード生成
 - コード生成のボタンをクリック

▼ コード生成とパッケージ化

コードの生成およびパッケージ化を行います。

プログラムの編集



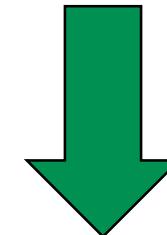
Python

- Pythonプログラムの編集手順
 - ConsoleConvert.pyの各関数を編集
 - `_init_`
 - 起動時によばれ、ポートの初期化を行う
 - `onActivated`
 - 非アクティブ状態からアクティブ化されるとき1度だけよばれる
 - `onDeactivated`
 - アクティブ状態から非アクティブ化されるとき1度だけよばれる
 - `onExecute`
 - アクティブ状態時に周期的によばれる

__init__ 関数の変更

- __init__ 関数内の InPort, OutPort の初期化を
以下のように変更する

```
self._d_InValue = RTC.TimedVelocity2D(*InValue_arg)  
self._d_OutValue = RTC.TimedFloatSeq(*OutValue_arg)
```



```
self._d_InValue = RTC.TimedVelocity2D(RTC.Time(0,0), RTC.Velocity2D(0.0, 0.0, 0.0))  
self._d_OutValue = RTC.TimedFloatSeq(RTC.Time(0,0), [])
```

RTC.TimedFloatSeq型に関して

- RTC.TimedFloatSeq型は構造体で以下のように定義されている

データ型	変数名	意味
Float[] (リスト)	data	float型のリスト
RTC.Time	tm	タイムスタンプ

- RTC.Time型は構造体で以下のように定義されている

データ型	変数名	意味
Int	sec	秒
Int	nsec	ナノ秒

RTC.TimedFloatSeq型について

- 使用例

```
self._d_OutValue.data = [0.0, 0.0] # 初期化  
self._d_OutValue.data[0] = 2          # 値の入力
```

onActivated関数全文

- onActivated関数を以下のように編集する

```
def onActivated(self, ec_id):  
    print "onActivated"  
    self._d_OutValue.data=[0.0, 0.0] # 初期化  
    return RTC.RTC_OK
```

onDeactivated関数全文

- onDeactivated関数を以下のように編集する

```
def onDeactivated(self, ec_id):  
    print "onDeactivated"  
    return RTC.RTC_OK
```

onExecute関数 | Importから値を読み込む

- 新しい値を読み込む手順

```
if self._InVelIn.isNew():      # 新しい値を受信しているか確認
    vel = self._InVelIn.read() # 新しい値を読み込む
```

- **isNew()** : 新しい値を受信しているか確認する
- **read()** : 新しい値を読み込む
 - 読み込んだ値は戻り値として格納する
 - 新しい値がない状態でread()した場合、値が存在しないためエラーになる
 - read()の前には必ずisNew()で値を受信しているか確認する必要がある

onExecute関数 | 各タイヤの角速度の計算

- 使い各タイヤの角速度を計算

```
# InPortからの値を使い各タイヤの角速度を計算  
wl = (vel.data.vx - vel.data.va * self._width[0]) / self._radius[0]  
wr = (vel.data.vx + vel.data.va * self._width[0]) / self._radius[0]  
print "wl:" + str(wl) + ",wr:" + str(wr)
```

コンフィギュレーション

- RTCBUILDERで宣言したコンフィギュレーションの変数は以下の名前で使用できる

コンフィギュレーション変数名	self._変数名
----------------	-----------

- このコンポーネントの場合、以下のようになる

コンフィギュレーション変数名	プログラム内での変数名
width	self._width
radius	self._radius

- 値の変更はプログラム実行時に、RTSystemEditor上でする

onExecute関数 | OutPortから値の出力

- 角速度を出力

```
# タイヤの角速度をOutPortの変数に入力後出力  
self._d_OutValue.data[0] = WL  
self._d_OutValue.data[1] = WR  
self._OutAngularVelout.write()
```

onExecute関数全文

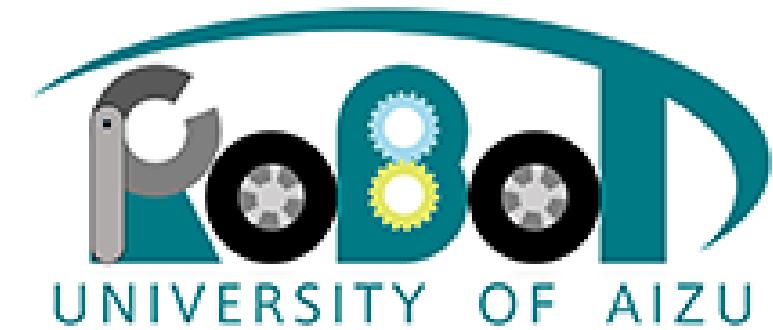
- onExecute関数を以下のように編集する

```
def onExecute(self, ec_id):
    # 新しい値を受信しているか確認
    if self._InVelIn.isNew():
        # 新しい値を読み込む
        vel = self._InVelIn.read()
        Wl = (vel.data.vx - vel.data.va * self._width[0]) / self._radius[0]
        Wr = (vel.data.vx + vel.data.va * self._width[0]) / self._radius[0]
        print "Wl:" + str(Wl) + ",Wr:" + str(Wr)
        # タイヤの角速度をOutPortの変数に入力後出力
        self._d_OutValue.data[0] = Wl
        self._d_OutValue.data[1] = Wr
        self._OutAngularVelOut.write()

    return RTC.RTC_OK
```

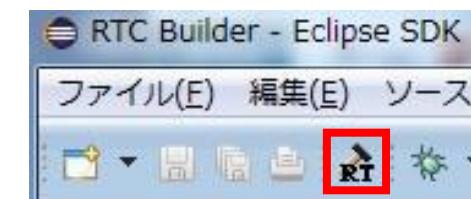
ConsoleOutAngularVel

コンポーネントの作成

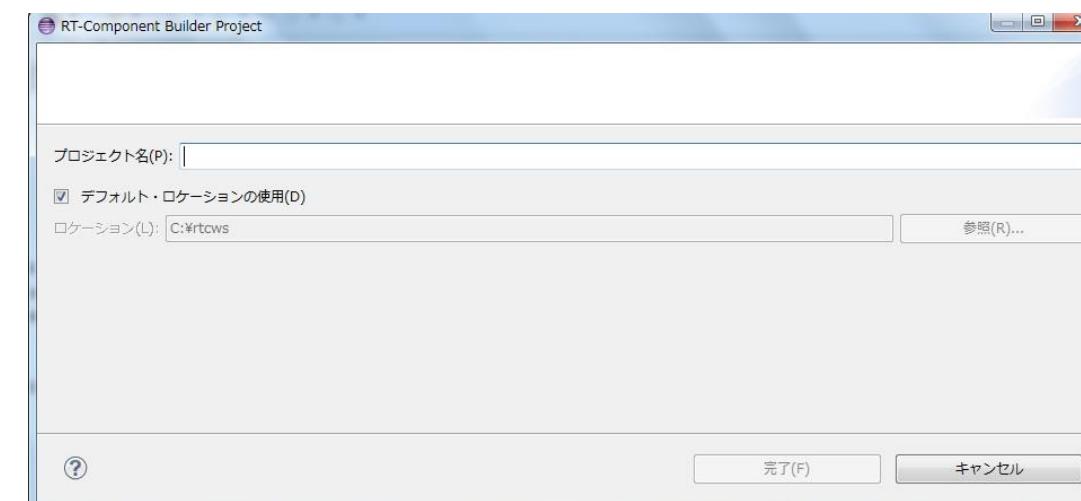


プロジェクトの作成

- ツールバー内の[RTとカナヅチ]のアイコンをクリック



- プロジェクト名を記入 (**ConsoleOutAngularVel**)



基本プロファイル

- コンポーネント基本情報の設定

モジュール名	ConsoleOutAngularVel
モジュール概要	任意(ConsoleOutAngularVel component)
バージョン	1.0.0
ベンダ名	任意
モジュールカテゴリ	任意(Category)
コンポーネント型	STATIC
アクティビティ型	PERIODIC
コンポーネントの種類	DataFlow
最大インスタンス数	1
実行型	PeriodicExecutionContext
実行周期	1000.0

基本

▼ RT-Component Basic Profile

このセクションではRTコンポーネントの基本情報を指定します。

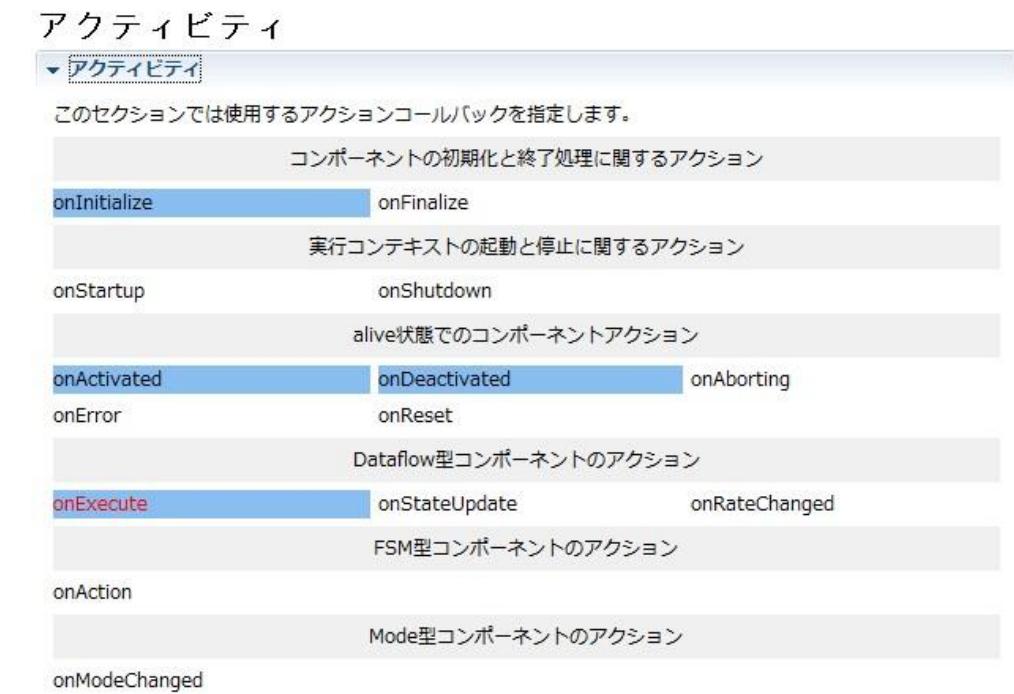
*モジュール名 :	ConsoleOutAngularVel
モジュール概要 :	ConsoleOutAngularVel component
*バージョン :	1.0.0
*ベンダ名 :	VenderName
*モジュールカテゴリ :	Category
コンポーネント型 :	STATIC
アクティビティ型 :	PERIODIC
コンポーネント種類 :	<input checked="" type="checkbox"/> DataFlow <input type="checkbox"/> FSM <input type="checkbox"/> MultiMode
最大インスタンス数 :	1
実行型 :	PeriodicExecutionContext
実行周期 :	1000.0

アクティビティ・プロファイル

- コンポーネントで使用予定のアクションコールバックを設定
 - 使用するアクションをダブルクリック（青背景にすることで選択する）
 - ※クリック1回目で文字が赤くなり、クリック2回目で背景が青くなる

選択アクションコールバック

- **onInitialize**
- **onActivated**
- **onExecute**
- **onDeactivated**



データポート・プロファイル

- RTコンポーネントのデータポートを設定

- 追加したいポートの[Add]ボタンを押し、ポートを追加し、直接名称を変更

*ポート名 (InPort)	
InAngularVel	<input type="button" value="Add"/>
	<input type="button" value="Delete"/>

- ポートのデータ型と変数名を入力

*データ型	RTC::TimedFloatSeq
変数名	InValue
表示位置	LEFT

データポート

▼ DataPortプロファイル
このセクションではRTコンポーネントのDataPort(データポート)の情報を設定します。

*ポート名 (InPort)	<input type="button" value="Add"/>	*ポート名 (OutPort)	<input type="button" value="Add"/>
	<input type="button" value="Delete"/>		<input type="button" value="Delete"/>

▼ Detail
このセクションではデータポート毎の概要を説明するドキュメントを記述します。
上のデータポートを選択すると、それぞれのドキュメントが記述できます。

ポート名 :

*データ型 : RTC::Acceleration2D
変数名 :
表示位置 : LEFT

Documentation

概要説明 :

データ型 :
データ数 :

意味 :

単位 :

発生頻度、周期 :

処理頻度、周期 :

基本 アクティビティ データポート サービスポート コンフィギュレーション ドキュメント生成 言語・環境 RTC.xml

InPort

- ポート名 : InAngularVel
- データ型 : RTC::TimedFloatSeq
- 変数名 : InValue
- 表示位置 : LEFT

サービスポート・プロファイル

- RTコンポーネントのサービスポートを設定
 - このコンポーネントでは使用しないので、スキップする

サービスポート



The screenshot shows the configuration interface for an RT-Component Service Port Profile. On the left, there is a panel titled "RT-Component Service Ports" containing a single entry: "sv_name". On the right, there is a detailed configuration panel titled "RT-Component Service Port Profile". It includes fields for "Port Name" (set to "sv_name") and "Display Position" (set to "LEFT"). Below these are sections for "Documentation", "Summary Description", and "I/F Summary Description", each with a large text input area.

RT-Component Service Ports

sv_name

Add Port
Add Interface
Delete

RT-Component Service Port Profile

このセクションではRTコンポーネントのServicePortの情報を設定します。

*ポート名 : sv_name

表示位置 : LEFT

Documentation

概要説明:

I/F概要説明 :

コンフィギュレーション・プロファイル

- RTコンポーネントのコンフィギュレーションを設定
 - このコンポーネントでは使用しないので、スキップする

コンフィギュレーション・パラメータ

▼ RT-Component Configuration Parameter Definitions

このセクションではRTコンポーネントのコンフィギュレーション・パラメータを指定します。

*名称		Add
		Delete

▼ Detail

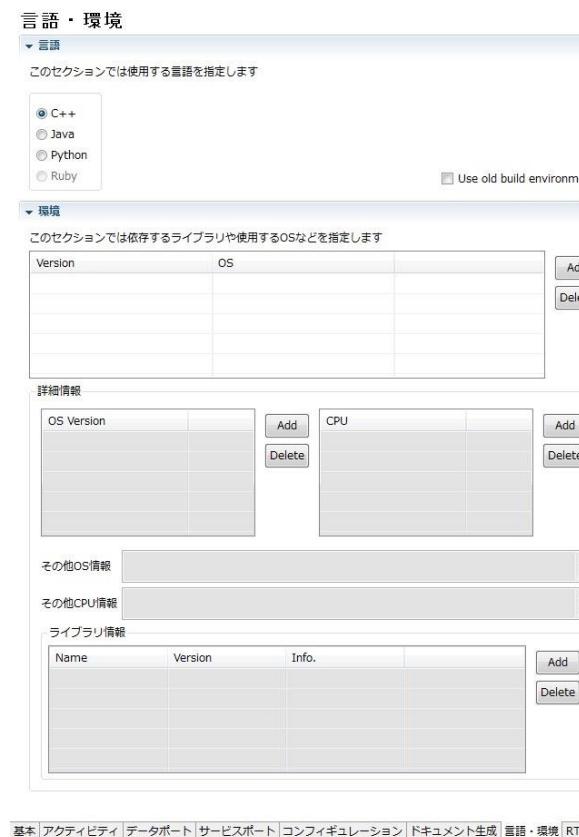
このセクションでは各コンフィギュレーション・パラメータの詳細情報を指定します。

パラメータ名 :

*データ型	short
*デフォルト値	
変数名 :	
単位 :	
制約条件:	
Widget:	text
Step:	

言語・環境・プロファイル

- RTコンポーネントの実装する言語、動作環境に関する情報を設定



Pythonを選択

コード生成

- 設定内容を確認

基本

- モジュール名 : ConsoleOutAngularVel
- モジュール概要 : 任意(ConsoleOutAngularVelcomponent)
- バージョン : 1.0.0
- ベンダ名 : 任意
- モジュールカテゴリ : 任意(Category)
- コンポーネント型 : STATIC
- アクティビティ型 : PERIODIC
- コンポーネントの種類 : DataFlow
- 最大インスタンス数 : 1
- 実行型 : PeriodicExecutionContext
- 実行周期 : 1000.0

選択アクションコールバック

- onInitialize
- onActivated
- onExecute
- onDeactivated

言語・環境

- Python

InPort

- ポート名 : InAngularVel
- データ型 : RTC::TimedFloatSeq
- 変数名 : InValue
- 表示位置 : LEFT

- すべての設定を終えたらコード生成
 - コード生成のボタンをクリック

▼ コード生成とパッケージ化

コードの生成およびパッケージ化を行います。

プログラムの編集



Python

- Pythonプログラムの編集手順
 - ConsoleOutAngularVel.pyの各関数を編集
 - `_init_`
 - 起動時によばれ、ポートの初期化を行う
 - `onActivated`
 - 非アクティブ状態からアクティブ化されるとき1度だけよばれる
 - `onDeactivated`
 - アクティブ状態から非アクティブ化されるとき1度だけよばれる
 - `onExecute`
 - アクティブ状態時に周期的によばれる

__init__関数の変更

- __init__関数内のInPort, OutPortの初期化を
以下のように変更する

```
self._d_InValue = RTC.TimedFloatSeq(*InValue_arg)
```



```
self._d_InValue = RTC.TimedFloatSeq(RTC.Time(0,0), [])
```

onActivated関数全文

- onActivated関数を以下のように編集する

```
def onActivated(self, ec_id):  
    print "onActivated"  
    return RTC.RTC_OK
```

onDeactivated関数全文

- onDeactivated関数を以下のように編集する

```
def onDeactivated(self, ec_id):  
    print "onDeactivated"  
    return RTC.RTC_OK
```

onExecute関数全文

- onExecute関数を以下のように編集する

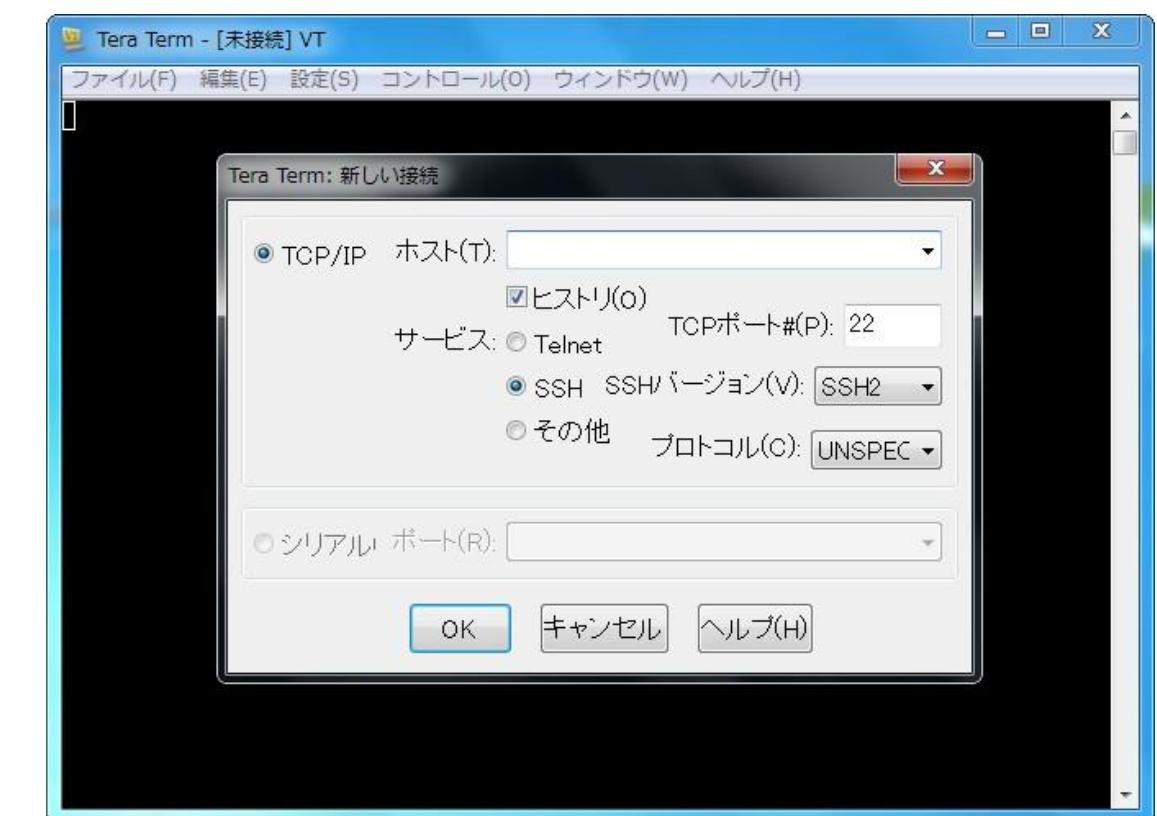
```
def onExecute(self, ec_id):  
    # 新しい値を受信しているか確認  
    if self._InAngularVelIn.isNew():  
        # 新しい値を読み込む  
        AngularVel = self._InAngularVelIn.read()  
        # 各タイヤの角速度を表示  
        print "Wr:"+str(AngularVel.data[1])  
        print "Wl:"+str(AngularVel.data[0])  
    return RTC.RTC_OK
```

Raspberry Piにコンポーネントをコピー

- フォルダをzip形式で圧縮
 - Raspberry Piにコピーしたいフォルダ(**ConsoleOutAngularVel**)をzipで圧縮する

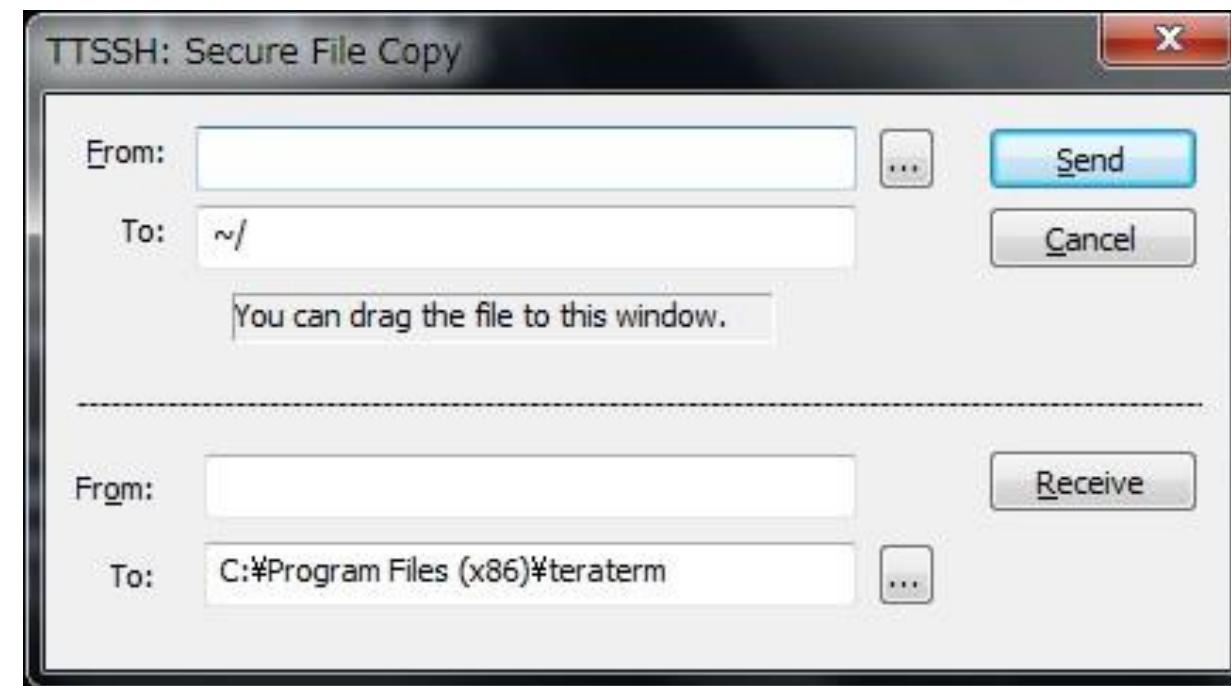
Raspberry Piにコンポーネントをコピー

- TeraTermでRaspberry Piに接続
 1. [ホスト名.local]または[IPアドレス]を入力
 2. IDとパスワード入力
 - ユーザー名: pi
 - パスフレーズ: raspberry



Raspberry Piにコンポーネントをコピー

- SCPでファイル転送
 - 「SSH SCP …」を使用してファイルをコピー



Raspberry Piにコンポーネントをコピー

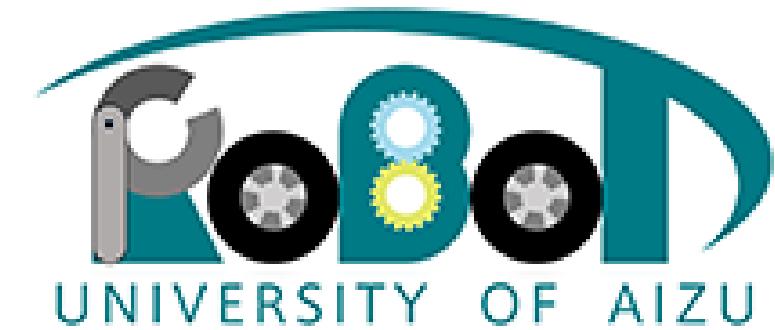
- コピーされたファイルを解凍する

```
$ unzip ConsoleOutAngularVel.zip
```

- 実行権限を変更する

```
$ cd ConsoleOutAngularVel  
$ chmod 777 ConsoleOutAngularVel.py
```

コンポーネントを接続



受講者PCでコンポーネント起動

- Naming Serviceの起動
 - [スタート]メニューから[プログラム]→[OpenRTM-aist x.y]→[tools]→[Start Naming Service]をダブルクリック
- ConsoleVellInコンポーネントの起動
 - ConsoleVellIn.pyをダブルクリック
- ConsoleConvertコンポーネントの起動
 - ConsoleConvert.pyをダブルクリック

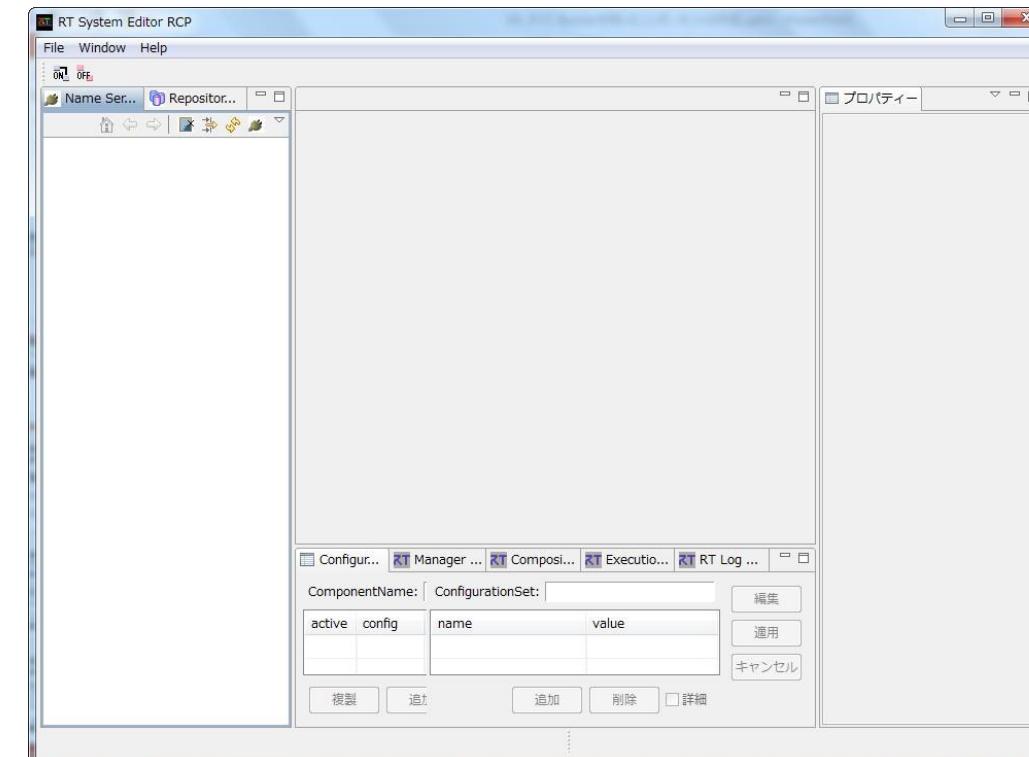
Raspberry Piでコンポーネント起動

- ネームサーバとコンポーネント起動

```
$ rtm-naming  
$ cd ~/ConsoleOutAngularVel/  
$ python ConsoleOutAngularVel.py
```

RTSystemEditor起動

- [スタート]メニューから[プログラム]→[OpenRTM-aist x.y]→[tools]→[RTSystemEditorRCP]を起動

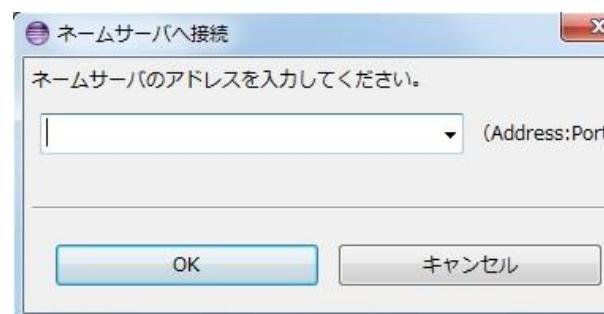


NameServer接続方法

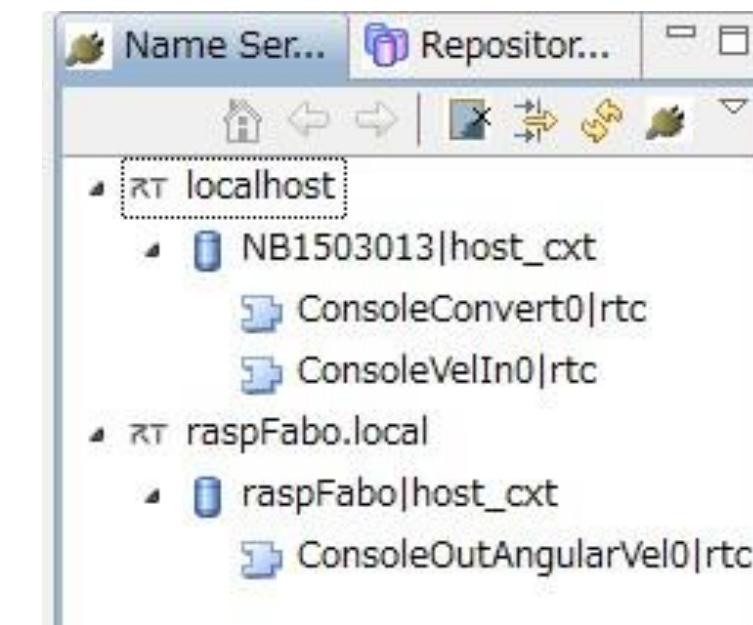
1. Name Service View のコンセントアイコンをクリック



2. ネームサーバのアドレスを聞かれるので、ホスト名かIPアドレスを記入

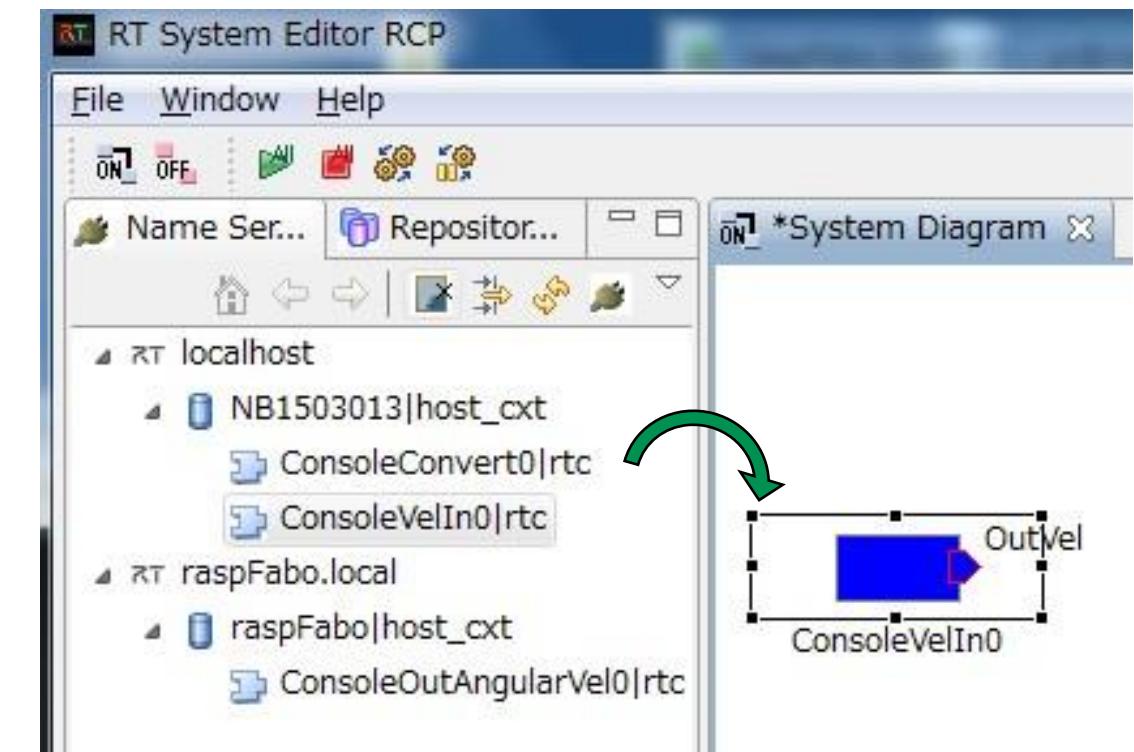


3. Name Service Viewに記入したものが記載すれば、接続完了



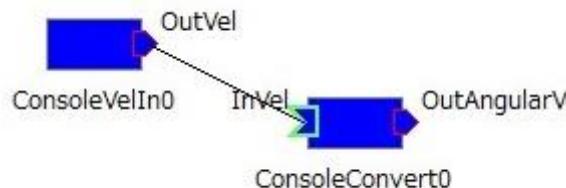
コンポーネント配置

- 対象コンポーネントを
ドラッグアンドドロップで、
System Diagram上に配置
- System Diagramが
開いてない場合は、
[ON]のボタンをクリック

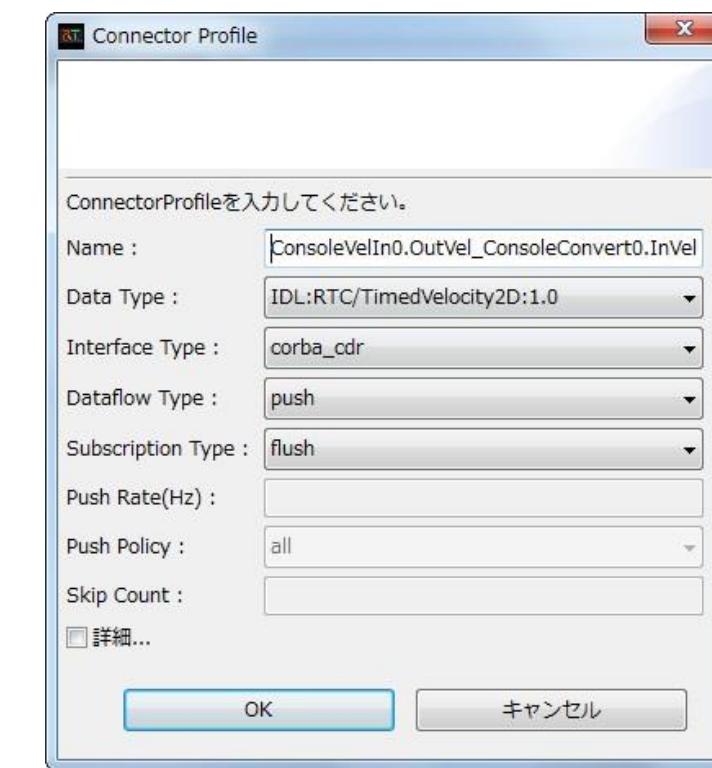


データポート接続

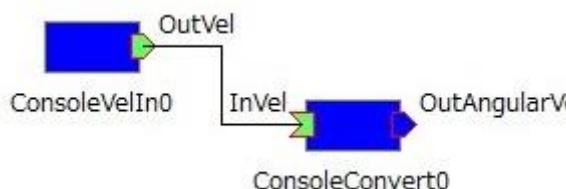
- 接続元のポートから接続先のポートまでドラッグアンドドロップ



- 接続プロファイルを記入してOKをクリック

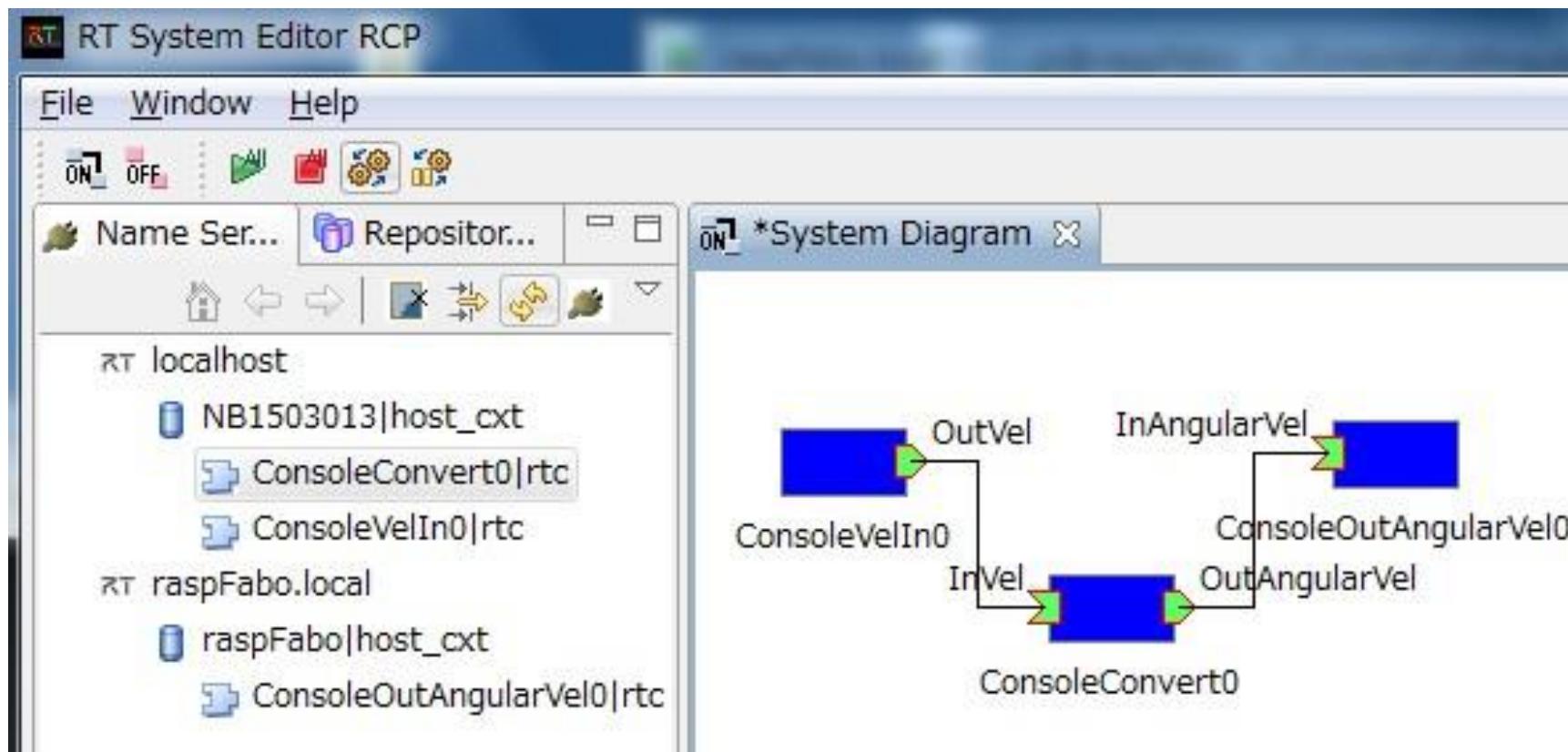


- 接続しているポートが緑色になれば完了



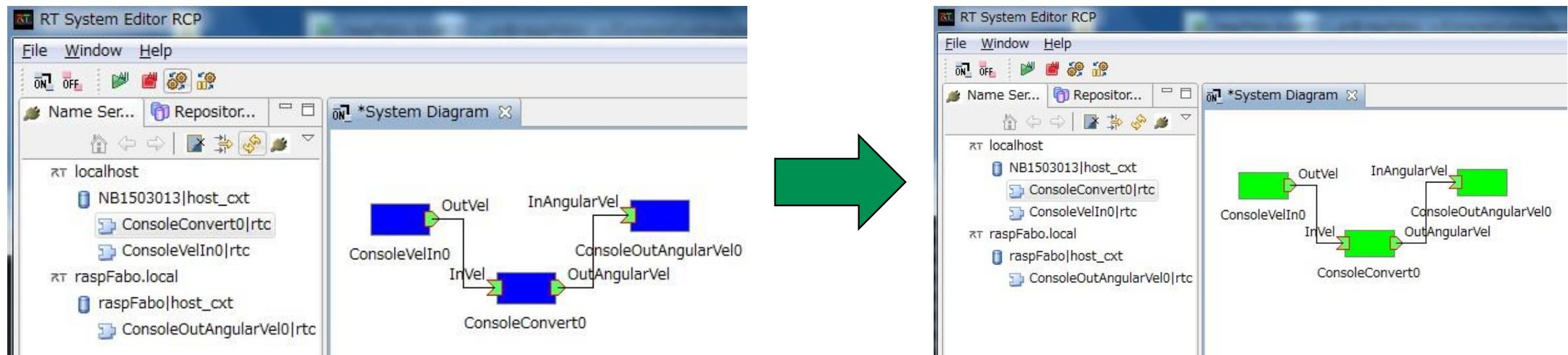
データポート接続

- データの型が合うように、以下の様に接続する



コンポーネントのアクティブ化

- コンポーネントのアクティブ化方法
 - 画面の緑の矢印をクリックする
 - または、右クリックから[All Activate]を選択する
 - コンポーネントが全て緑色になれば、アクティブ化完了



コンポーネント動作確認

- ConsoleVelIn : 2次元速度ベクトルを入力
- ConsoleConvert : 各値と計算式を表示
- ConsoleOutAngularVel : コンポーネントウィンドウに角速度を表示

コンポーネント動作例

- Vxに1, Vaに0を代入したときの表示

```
C:\Python27\python.exe
onActivated
input Vx:
1
Vx:1
input Va:
0
Va:0
input Vx:
```

ConsoleVelInでの表示

```
C:\Python27\python.exe
onActivated
WI:36.3636363636,Wr:36.3636363636
```

ConsoleConvertでの表示

```
ET15.local:22 - pi@ET15: ~/ConsoleoutAngularVel \
[ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W)]
pi@ET15:~/ConsoleoutAngularVel $ python C
Traceback (most recent call last):
  File "/usr/lib/python2.7/dist-packages/raspirobotlib/Robot.py", line 3, in <module>
    from . import config
      ^~~~~~
  File "/usr/lib/python2.7/dist-packages/raspirobotlib/config.py", line 3, in <module>
    from . import prop
      ^~~~~~
  File "/usr/lib/python2.7/dist-packages/raspirobotlib/prop.py", line 3, in <module>
    from . import load
      ^~~~~~
  File "/usr/lib/python2.7/dist-packages/raspirobotlib/load.py", line 3, in <module>
    if _str[len(_str)-1] == "##" and not _str[-2].isalpha():
                                         ^
IndexError: string index out of range
onActivated
Wr:36.3636360168
WI:36.3636360168
```

ConsoleOutAngularVelでの表示