

Visual Components Advanced

TM670



贝加莱工业自动化
Perfection in Automation
www.br-automation.com



前提

培训模块: TM610 – ASiV基础

软件: AS 2.5 或更高版本/AR 2.80 或更高版本

硬件: 4PP220.1043-75

目录

1 · 介绍	3
1.1 目标	4
2 · VISAPI 编程接口	5
2.1 VISAPI 库	6
2.2 VISAPI 运行函数	6
2.3 DrawBox 控制	9
3 · 运行性能	13
3.1 可视化组件的运行行为	13
3.2 运行按键动作	14
4 · 操作和输入行为	15
4.1 按键优先级	15
4.2 复合键动作	17
4.3 密码控制	20
4.4 切换键盘布局	21
5 · 动态处理图像元素	26
5.1 动态设计控制颜色	26
5.2 用状态数据点操作控制	27
5.3 文本段	28
5.4 输入锁定	31
6 · 内部数据源	32
7 · 总结	33

1. 介绍

可视化应用程序必须能够满足更高的要求，而不仅仅是显示文本、过程值或图像之类的需求。在目标系统中也可以设计更为复杂的可视化应用。

我们为可视化组件用户提供了可编程工具，可以用它来改变过程显示。

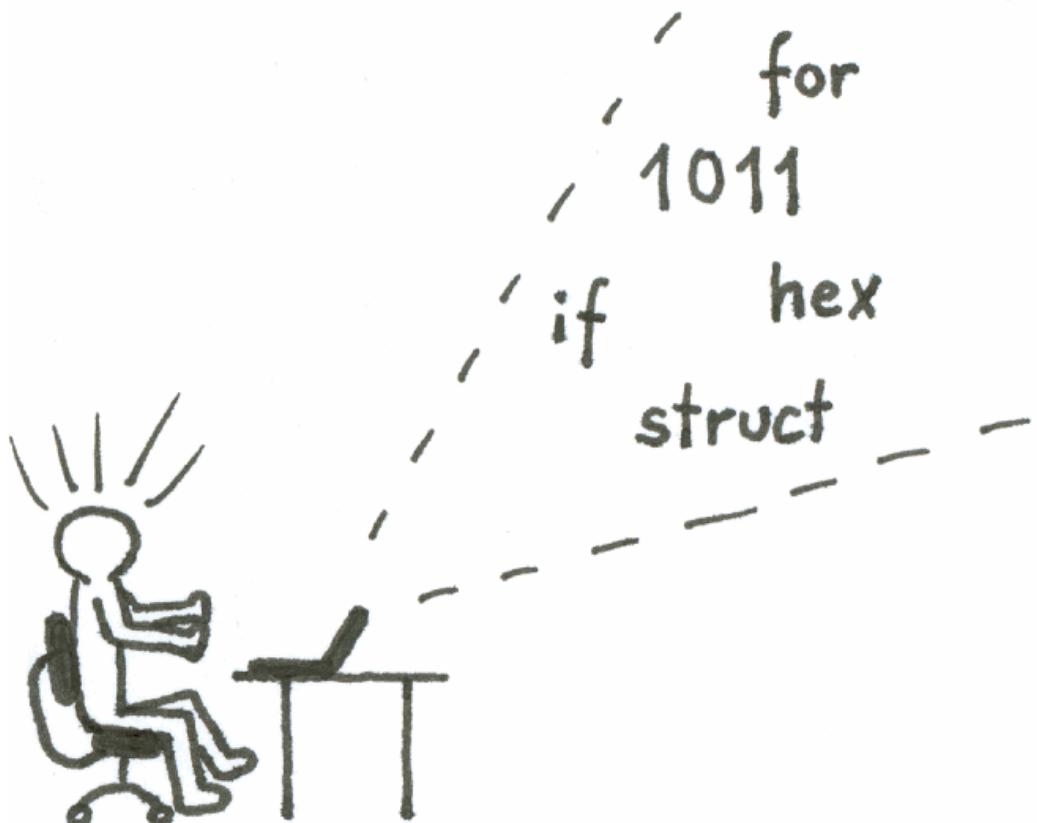


图1 高级可视化组件

这个培训模块用各个相互联系的主题来描述当使用可视化组件时可用哪些应用程序。

在使用这个培训模块之前，必须对可视化组件编辑器和可视化控制有一个直观的理解。

应用可视化组件培训模块 TM610、TM640或TM650中的项目来实现各个练习和任务。

目录

1.1 目标

能够完成复杂的可视化任务。

在这个培训模块中，我们能够了解到通过可视化组件可以执行什么样的高级“操作”和“监控”。

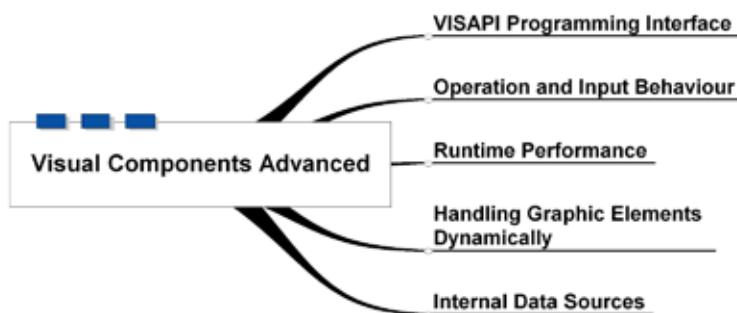


图2 综述

2、VISAPI 编程接口

可视化组件编程接口, "VISAPI" 库, 允许用户在运行时输出应用程序中的图像和文本。

复杂的动态图像的显示也可以像简单的显示功能, 如调节显示的对比度和亮度那样来调节。

帮助仍然是本章中的一个重要工具。



按<F1>键, 从库管理器中启动VISAPI 帮助 (如其它库一样)。

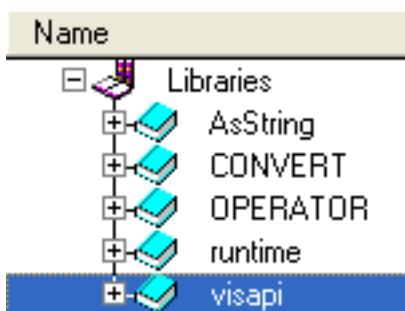


图3 库管理器 – VISAPI

注意

这个培训模块没有描述每个VISAPI 功能的配置, 而是通过具体的任务来讨论这些功能的使用。通过VISAPI库中的程序实例来说明各个功能的使用。

2.1 VISAPI 库

当在AS中创建一个新的可视化应用程序之后，VISAPI 库会自动添加到项目中，并可以立刻应用到应用程序中。

VISAPI 可以在以下任务中应用：

- 输出图像和文本
- 设置对比度和亮度
- 读取按键矩阵（同样参看4.1按键优先权）
- 校准触摸屏并确定触摸位置
- 读取报警列表

2.2 VISAPI 运行函数

没有 VISAPI 函数，Visual Components Runtime 可以处理图像结构和按键评估。

当执行VISAPI 函数时，VC Runtime 也可以控制屏幕上显示相关功能的任务。

2.2.1 连接可视化应用程序

通过函数库VISAPI中的 函数VA_Setup可以连接可视化应用程序，这个函数返回一个句柄，当使用其它任何一个VISAPI函数时，需要将这个句柄传送。

由于在目标系统中（本地 / 远程）可以运行多个可视化应用程序，所以可以通过可视化任务的名字来选择可视化应用。

注意

需要循环调用 VA_Setup 函数，直到函数返回一个有效的句柄（值不为0）。

每次循环只调用一次函数。如果返回0值，那么必须在下个循环周期中再次调用函数。

问题

VA_Setup 函数能够由多个用户任务或者用户函数调用么？

答案

是。每次调用 VA_Setup 函数时会返回相同的句柄。

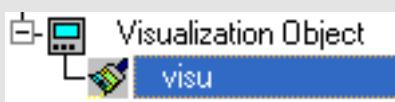


任务： 在应用程序中使用 VA_Setup 函数

在用户任务中，可以执行 VA_Setup 函数。VISAPI 帮助中为每个函数提供了 "Automation Basic" 或者 "ANSI C" 的编程实例。

方法

在 VA_Setup 函数中，可视化对象的名称必须在输入参数的“注释”中定义。



注意

可视化对象的名字在输入时要区分大小写。

结果

值不为 0 = 必须返回句柄。现在，可以使用其它所有的 VISAPI 函数了。

2.2.2 调用 VISAPI 函数

Visual Components Runtime 既要处理VC 编辑器画的可视化对象的屏幕显示又要处理VISAPI 函数的显示。

这样，屏幕显示必须同步。

VISAPI 函数访问是通过函数VA_Saccess 和 VA_Srelease 来控制的。

例— 使用函数 VA_Saccess 和 VA_Srelease

```
(* cyclic program *)  
  
if ready <> 1 then  
    VC_HANDLE = VA_Setup(1 , "visu")  
  
    if VC_HANDLE <> 0 then  
        ready = 1  
    endif  
endif  
  
if ready = 1 then  
    if VA_Saccess(1,VC_HANDLE)= 0 then  
        ....  
        VA_Srelease(1,VC_HANDLE)  
    endif  
endif
```

重要

只有当函数VA_Saccess返回status = 0时，才能执行VA_Srelease 函数。
确保不能忘掉函数 VA_Srelease ，也不会延迟函数调用。

很多 VISAPI 画图函数需要在这两个函数中输出。

2.3 DrawBox 控制

如果在没有DrawBox控制的情况下来看 VISAPI 函数的应用，那么VISAPI 画图命令会输出到函数指定的 x 和 y 位置（显示画面的左上角）。

这样做，用户必须考虑下面几项：

- 用户必须在VC可视化对象中的对象可视化控制和VISAPI画图函数之间定义同步。
- 当打开一个触摸板时，画图函数不能在背景上显示——用输入区域的状态变量可以实现同步。
- VISAPI 函数 VA_Redraw 可以清除屏幕上所有的 VISAPI 输出（除了DrawBox 控制）。

这意味着每个屏幕事件必须通过应用程序的确认，而且为了图片结构必须重新调用VISAPI 函数。

一般不建议不经过DrawBox控制而从VISAPI使用显示的方法。

DrawBox 可以理解为一个“显示中的显示”。Visual Components 编辑器中的图像区域在 VISAPI 函数要显示的地方来定义。

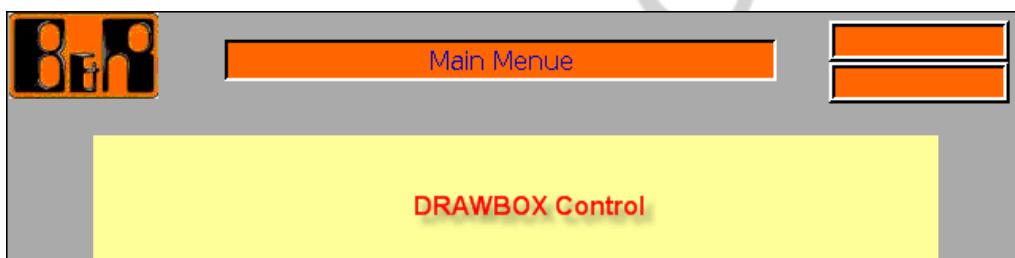
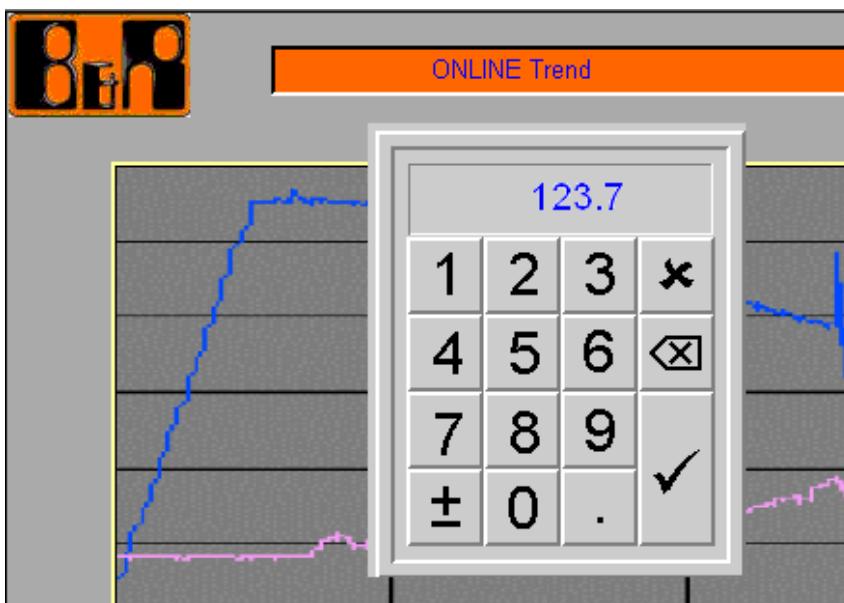


图4 放置 DrawBox 控制

DrawBox 具有以下优点：

- 绘图函数在DrawBox 控制的左上角输出。
- 重新放置DrawBox 控制不会影响用户任务。
- 当打开一个触摸板时，绘图函数根据Z-Order在背景中的DrawBox 中绘制。即使VC对象的可视属性改变了，输出也会自动刷新。



- 图像每250ms会自动刷新。使用VISAPI 函数VA_Redraw 重新绘制DrawBox。
- 图表可以用DrawBox 状态变量显示或者隐藏。

注意

图像说明可以使用VISAPI 函数显示趋势的可能性。这个趋势函数并不是可视化组件的特性。

使用the DrawBox时以下几点应该注意：

- 在换页和切换语言时，DrawBox 的内容会消失。
- 不能使用透明背景（已增加的执行要求）。
- VA_Redraw 强制刷新 DrawBox。必须绘制充满所需背景颜色的矩形来清除所有的 VISAPI 绘图函数。

2.3.1 访问 DrawBox控制

当调用DrawBox控制时， VISAPI 绘图函数按照默认直接在显示中绘制。

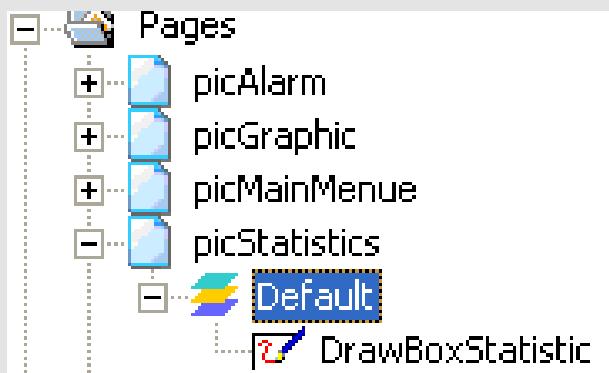
在DrawBox控制更改输出时，需要使用VISAPI 函数 VA_Attach 。

这里也一样，只有当函数返回status = ZERO时，才能使用绘图函数。

在可视组件的应用中，不能在一页或多页中使用多个 DrawBox 控制。通过指定 VA_Attach 函数的"Connection"名字来控制访问。

例: 在一页中访问 Drawbox 控制

在下面的例子中，DrawBox 控制放置在页面的默认层。



传送到VA_Attach函数中的连接字符串具有以下结构:

PageName/LayerName/ControlName

根据以上的图像做一个连接:

"picStatistics/Default/DrawBoxStatistic"

如果将DrawBox放置在全局层上，那么连接字符串具有以下结构:

Layername/Controlname

2.3.2 中止访问 DrawBox 控制

在VA_Attach之后所有的 VISAPI 函数调用都应用到这个DrawBox。

必须先中止对当前 DrawBox 控制的访问，才能再次访问整个屏幕页面或另一个 DrawBox 控制。

这要用到 VA_Detach 函数。

任务：从 DrawBox 文本组中输出一个文本



Access 用 VISAPI 库的访问应该建立在一个存在的用Visual Components 的 Automation Studio工程基础上。由于每个库函数都有程序实例，VISAPI库帮助能够用来实现这个任务。

现在必须执行以下步骤：

- 在屏幕页面中插入一个 DrawBox 控制。
- 在用户任务中，通过VA_Setup函数读取可视化对象的句柄。
- 一旦收到有效的句柄，就可以通过DrawBox 控制建立连接了（不要忘记 VA_Saccess, VA_Srelease函数）。
- 通过VA_GetTextByTextGroup 函数，在Visual Components 编辑器中从使用的文本组中读取文本。
- 通过VA_Textout将文本输出到屏幕上。

注意

培训模块 TM610、TM640或者 TM650中的项目可以作为一个基础。在培训项目TM650中， 可以监控VISAPI函数的语言切换效果。

3、运行性能

有必要了解可视化组件的运行过程，从而执行可视化和运行的时间临界任务。

3.1 可视化组件的运行行为

在 Automation Runtime 的目标系统中，系统在空闲时间会运行可视化组件的可视化应用。

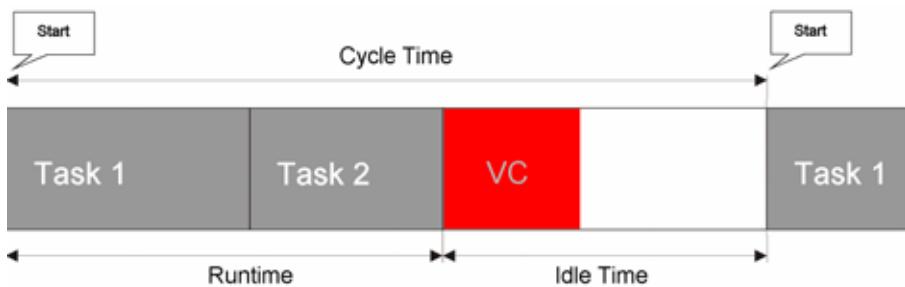


图5 可视化组件 – 运行行为

由于循环过程不受可视化程序的影响，所以只有有限的运行资源可以用到可视化组件中。

可视化组件的可视应用处理性能依赖于运行系统的可用空闲时间。

注意

关于 Automation Runtime 系统的详细信息可以在培训模块 TM213 中找到。

3.2 运行按键动作

每个按键的敲击（“硬件”键及“触摸”按钮或者热区）都存储在一个矩阵中。可视化组件“键盘驱动器”会处理这个矩阵中的按键信息，这些信息包含了缓存器中每个按键按下（1）或者释放（0）的信息：

- 按键按下时的时间
- 按键松开 / 重复按键 / 按键按下事件

VC Runtime从缓存器中读取信息并执行全局和局部的（在当前屏幕页面中配置的）按键动作：

- 当改变某页时，用比执行时间长的时间段检查缓存器中输入的按键动作，必要的话放弃该处理。
- 执行各个虚拟按键（VK）的动作。
- 全局按键矩阵的数据点通过键盘图像写入。

重要：

由于在注释的上下文中处理配置在VC 编辑器中的按键，所以可能会有不同的响应时间。在这种情况下，不能保证实时动作。

如果需要，那么必须使用VISAPI 函数 VA_GetKeyMatrix。（参考4.1）。

4. 操作和输入行为

系统操作对可视化提出了一些挑战。Visual Components 编辑器已经包含了所需的大部份解决方案。未提供的解决方案可以通过创建用户任务来处理。

- 实时按键处理
- 一键按下时的多个按键功能动作
- 密码输入和权限级别
- 切换键盘布局

4.1 按键优先级

我们在前面讨论得知，在VC 运行中处理配置在可视化组件中的按键动作和按键矩阵。

为了得到“实时”按键响应，任务级别内容中的按键矩阵可以通过VISAPI 函数 VA_GetKeyMatrix 来处理。

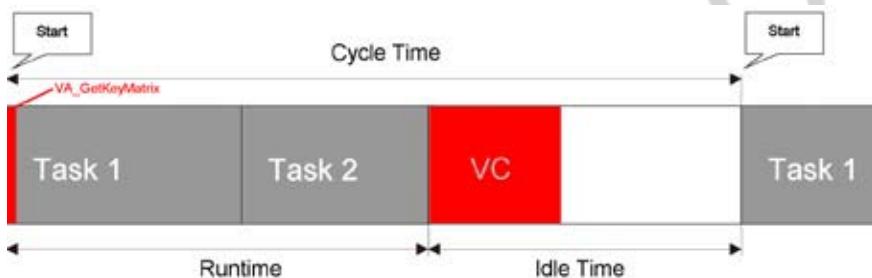


图6 运行时间 – 按键矩阵优先级

这意味着当前按键按下（1）和释放（0）的图像可以在所需的任务级别中得到。

任务级别优先级在AS项目硬件树中的“Display”部分设定。
在右边对话框中可以为各个可视化对象改变任务级别的优先级。

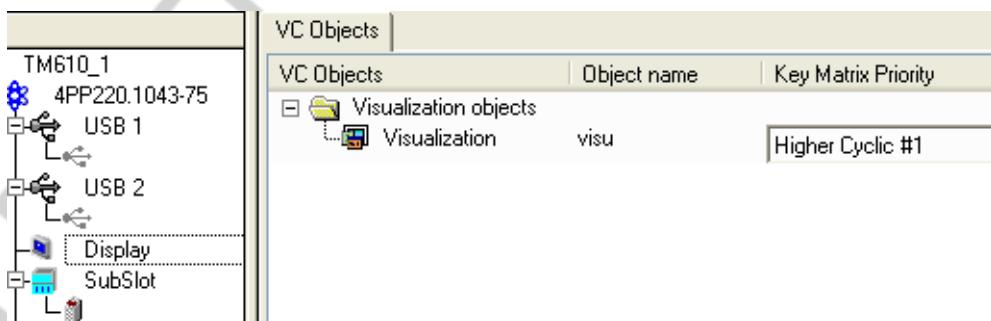


图7 设置按键矩阵优先级

操作和输入行为

4.1.1 为按钮和热区定义按键矩阵

实际按键根据它们的编号自动地在按键矩阵中显示。

按钮和热区的区别：如果在按键矩阵中也要显示按钮和热区，那么必须在属性中定义按键矩阵中的偏移量。

在按钮或者热区控制的"MatrixOffset"属性中，可以定义任何正数。

当通过硬件按键和触摸屏使用显示时，这个偏移量用来定位硬件按键。

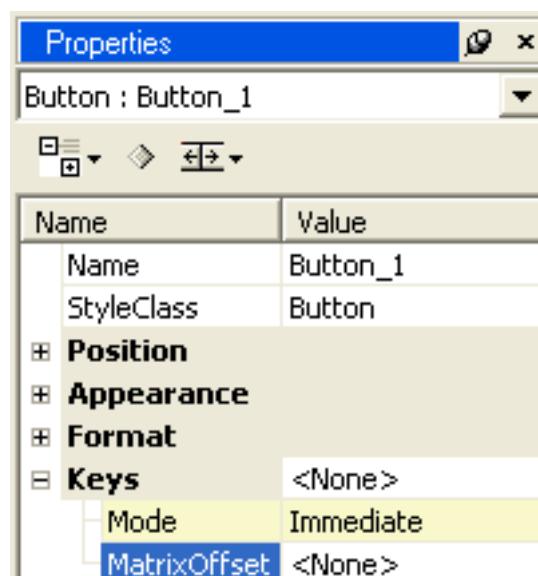


图8 按键矩阵偏移量

Hardware keys	+0	+1	+2	
0 ↑ 23	24	25	26	

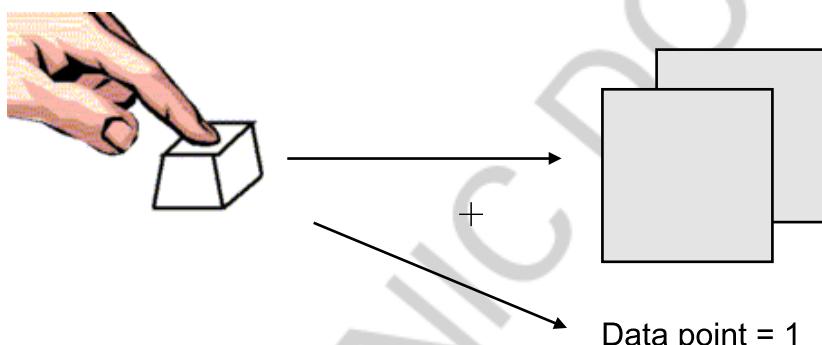
按键矩阵索引号 (数组索引号)

4.2 复合键动作



通常，分配给相关 VK的一个动作，比如换页，是由按键敲击或者触摸输入来执行的。

这个动作可以局部的或者全局的在虚拟按键（VK）执行。
如果一个VK必须同时执行多个动作（如换页和设定数据点），那么可以定义一个附加按键动作。



因此，也可以在VK中定义一个附加的全局动作或者局部动作，配置好的动作可以应用到整个可视化程序中。这个局部动作只在当前配置的页面中执行。

注意

当在一个按键上使用复合键动作时会有一些限制。这里有些“唯一”的按键动作，不允许其它任何动作。在Visual Components 编辑器（有限制的选择）中会禁止这些配置。

操作和输入行为

4.2.1 用两个全局动作创建一个 VK

在“按键”节点下，将所有的虚拟按键输入到Visual Components 编辑器中。

◆ VK_picMainMenue	ChangePage	Target(Page, picMainMenue)
-------------------	------------	----------------------------

如上所见，全局 VK "VK_picMainMenue" 执行一个换页功能。

另外，通过按下<Insert>键添加空动作。

◆ VK(picMainMenue)		
- ◆ Action_0	ChangePage	Target(Page, picMainMenue)
- ◆ Action_1	<no type>	

图9 插入全局虚拟按键

第一个配置动作显示为 "Action_0"。新动作可以在"Action_1"的属性列表中配置。

4.2.2 用一个全局和局部的动作创建一个 VK

局部动作在一个页面上配置。所有可用的 Vks 在编辑器的“Keys”表中显示。

Name ▲	Type	Details
◆ VK(picGraphic)		used globally
◆ VK(picMainMenue)		used globally
◆ VK(picTextValue)		used globally

Design | Layers | **Keys** | LEDs | Panel |

图10 图像编辑器中显示虚拟按键

按<Insert>键将局部动作添加到全局VK 中。

4.2.3 用两个局部动作创建一个VK

在编辑器中，局部VKS也可以在“Keys”表中显示动作。

Name	Type	Details
◆ VK_locSetPoint1	SetDatapoint	Value(<none>, 0)
◆ VK_picGraphic		used globally
◆ VK_picMainMenue		used globally
◆ VK_picTextValue		used globally

图11 图像编辑器中局部虚拟按键的显示

按<Insert>键添加一个附加的局部动作。

Name	Type	Details
◆ VK_locSetPoint1		
◆ Action_0	SetDatapoint	Value(ready, 0)
◆ Action_1	<no type>	
◆ VK_picGraphic		used globally
◆ VK_picMainMenue		used globally
◆ VK_picTextValue		used globally

图12 插入一个局部动作

操作和输入行为

4.3 密码控制

密码控制 可以用作隐藏输入（也就是密码输入）。

这个控制也可以用作用户管理（应用层）。

密码控制功能：

- 一个文本或数值的隐藏输入。
- 比较文本组中的文本输入或者用户任务中的字符串数组。
- 将输入的密码层写到相连的数据点上。
- 密码层的号码和输入允许的字符个数可以自由定义。



在属性列表中配置密码控制。

这个例子表明，密码在 "Passwords" 文本组中来管理。

第一个密码层的密码在 "TextIndexOffset" 属性中定义。

在输入一个有效的密码后，属性 "LevelDatapoint" 以“1”开始包含各个层。无效的密码由“0”来表示。

Name	Value
Name	Password_1
StyleClass	Password
Position	
Appearance	
Value	
Source	MultipleTexts
TextGroup	Passwords
TextIndexOffset	0 : superuser
Text	
LevelDatapoint	...!GLOBAL.PasswordLevel
MaxChar	<None>
MaxLevel	3
SimulationValue	
Input	
Start	Any Key
Confirm	Enter
Next	Disable
Cancel	Lost Focus
Mode	Password
PasswordChar	*
TouchPad	AlphaPad

图13 密码控制属性

4.4 切换键盘布局

切换键盘布局可以“去除”使用触摸板定义的限制。这是由语言切换（"WERTZ" 键盘 – "QWERTY" 键盘）或者显示分辨率（QVGA displays）决定的。

在建立一个触摸板时确定按键层的号码。

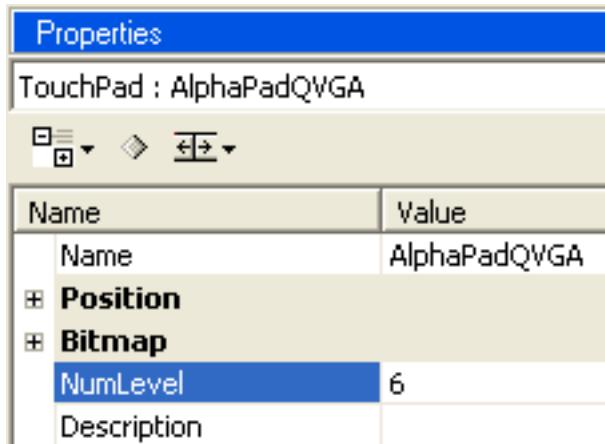


图14 触摸板 – 按键层

在标准项目中，已经为QVGA显示提供了一个键盘布局。用这个例子来说明切换键盘布局的配置。

必要条件：

- 各个层的所有位图必须可用。
- 为了从视觉上区分释放或者按下状态，在每个位图层上必须创建 按下和释放的位图。

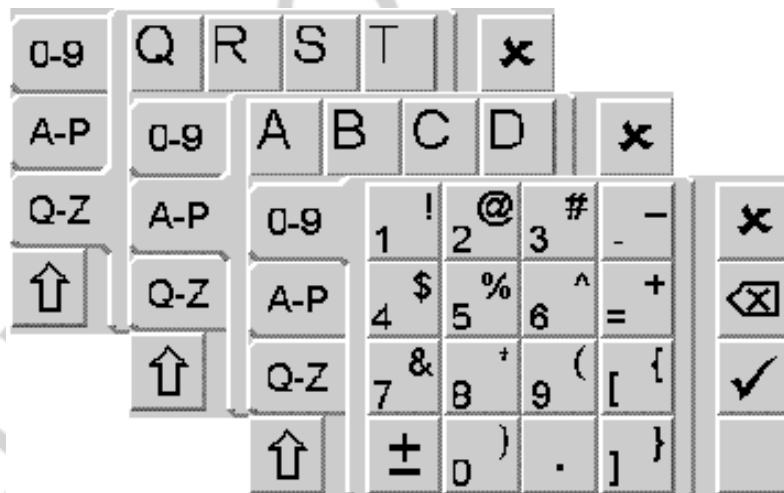


图15 QVGA 键盘 – 释放状态

操作和输入行为

我们的例子现在包括6个不同的位图，其中3个为释放状态另外3个为按下状态。这些位图将在单独的位图组中管理。

问题

为什么每个位图在表中输入两次？

答案

相同的位图在小写和大写字母时使用。

Index ▲	Bitmap
0	AlphaPadQVGA1
1	AlphaPadQVGA1
2	AlphaPadQVGA2
3	AlphaPadQVGA2
4	AlphaPadQVGA3
5	AlphaPadQVGA3
6	AlphaPadQVGA1_pressed
7	AlphaPadQVGA1_pressed
8	AlphaPadQVGA2_pressed
9	AlphaPadQVGA2_pressed
10	AlphaPadQVGA3_pressed
11	AlphaPadQVGA3_pressed

图16 QVGA 位图列表

4.4.1 插入一个触摸板

点击"Keys" 节点上的<Insert>按钮添加一个新的触摸板。

在这个触摸板的属性下，将位图组与释放和按下属性相连接。

释放和按下状态的位图是由使用"BitmapIndexOffset"来定义的。

Properties	
TouchPad : AlphaPadQVGA	
Name	AlphaPadQVGA
Position	
Bitmap	
Released	MultipleBitmaps
BitmapGroup	AlphaPadQVGA
BitmapIndexOff...	0
Pressed	MultipleBitmaps
BitmapGroup	AlphaPadQVGA
BitmapIndexOff...	6
NumLevel	6

图17 QVGA 触摸板属性

4.4.2 定义触摸栅格

各个位图的区域通过自由定义的触摸栅格为按键分配来分割。

这个栅格可以通过可变的高和宽分成行和列。这样，很容易将栅格调整到位图的尺寸。

注意：

当创建键盘位图时，确保这种分割允许的间隔可以被栅格覆盖。

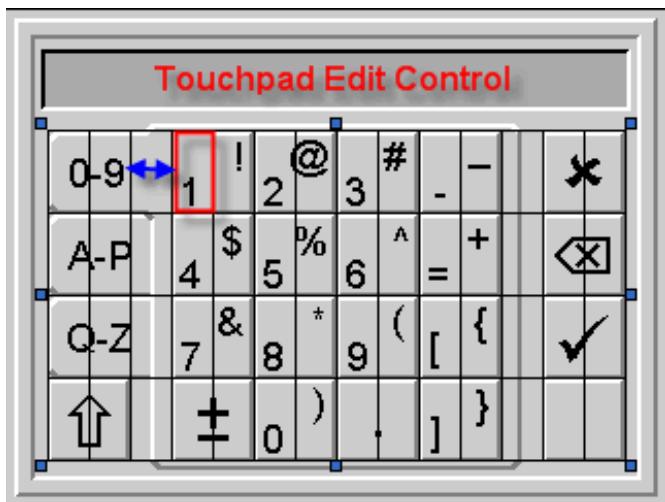


图18 触摸板 – 栅格

这样，你可以看到按键尺寸被两个栅格部分（红色矩形）覆盖。这些栅格部分的尺寸由各个区域间（蓝色箭头）的间隔来确定。

操作和输入行为

4.4.3 按键分配（映射）

将“按键编码”分配给“映射”表中触摸板栅格的一个元素。

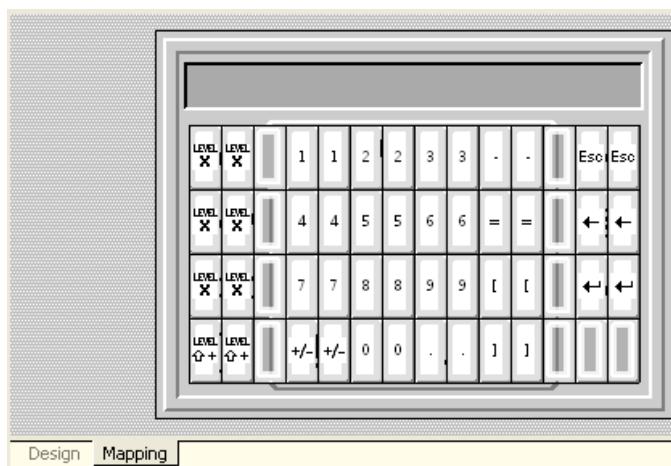


图19 按键分配

在这个图像中，你可以看到对每个按键必须执行两次分配。
哪种分配是必须的？

- 按键编码分配（为大小写字母分配数字和字母的字符）
- 控制按键分配（Enter, ESC, backspace等）
- 为大小写字母的层切换分配（索引号+1）
- 为位图切换的层切换分配（索引号+2）

按键层通过确定层号（为大小写字母的3位图 \times 2）会自动定义。

这些分配使用"Virtual Key[0]" 到 "Virtual Key [layer – 1]"来显示。
在每一层中必须插入一个新的虚拟按键，显示在 "Value" 一栏中。

Name	Value
Virtual Key[0]	TP_ALPHA_1
Virtual Key[1]	TP_ALPHA_Exclamation
Virtual Key[2]	TP_ALPHA_sa
Virtual Key[3]	TP_ALPHA_A
Virtual Key[4]	TP_ALPHA_sq
Virtual Key[5]	TP_ALPHA_Q

图20虚拟按键

在这个例子中，在按键"1", "A" 和 "Q"处为各个位图定义了虚拟按键：

- 第一层输出数字 "1"。
- 按Shift 键时输出惊叹号 "!"。
- 当切换到第二层时，输出字母"a"。
- 如果在第二层按下Shift键时，输出字母 "A"。
- 切换到第三层后，输出字母 "q"。
- 如果在第三层按下Shift键时，输出字母 "Q"。

Name	Value
Virtual Key[0]	TP_ALPHA_1
Action	
Type	InputCharacter
LanguageDepen...	False
Character	1

图21 按键分配

层切换:

通过动作"ChangeKeyLevel"以及设定"KeyLevel"来切换层。模式 "Set"用来确保 KeyLevel 保持在设定值。

Virtual Key[0]	TP_SET_LAYER0
Action	
Type	ChangeKeyLevel
Mode	Set
KeyLevel	0

Shift 键:

当按下Shift 键时，通过"ChangeKeyLevel"动作也可以切换层。但是，模式 "Increment Single" 定义了只能在敲击下一个按键时层才会切换。

动态处理图像元素

5. 动态处理图像元素

当设计屏幕页面时，会在开发过程中为相关控制给定一个静态外观。

图像元素可以表述如下：

- 改变控制的前景或者背景颜色
- 控制可视性能和操作
- 运行时编译文本
- 锁定输入

5.1 动态设计控制颜色

在开发过程中，将前景和背景颜色分配到图形控制中。

Appearance	
Border	<None>
ForeColor	<input type="color"/> 0
BackColor	<input type="color"/> 16
ColorData...	<None>

通过 "ColorDatapoint"任意连接16位的整数值。

如果一个数据点连接到这个属性，那么会在运行中覆盖控制中定义的颜色设置。

低8位 (Bits 0-7)作为背景颜色，高8位(Bits 8-15)用作前景颜色。

$$\text{ColorDatapoint} = \text{背景颜色} + \text{前景颜色} * 256$$

5.2 用状态数据点操作控制

状态数据点可以任意连接到任何图形元素上。

注意

可视化组件的帮助会指出哪些状态位能够用于每个控制。

状态位	访问	功能
0	W	Visibility : 此位设定后控制不可见，清除此位后控制重新可见。
1	W	Locking: 此位设定后，锁定控制输入。
2	W	Focus: 此位设定后控制设为中心。
3	W	Open touch pad: 此位设定后触摸板为输入区域打开。
12	R	IsLocked: 如果按下一个锁定的输入控制，那么通过VC设定此位。
13	R	Overlay is open: 如果打开触摸板，那么通过VC设定此位。
14	R	Control has focus: 如果控制拥有focus，设定此位。当控制不再有focus后，清除此位。
15	R	Control input: 编辑数值时，通过 VC 设定此位。当结束或取消输入时，重新设置此位。

重要

状态位 2 和 3: 在屏幕页面中，状态位只能为一种控制来设定，否则行为不明确。另外，这个位不能自动重设，用户必须自己小心这点(估计位 14 和 15)。

5.3 文本段

文本段在运行时用来动态地把文本群或者警告群中的一个文本连在一起。
文本群中的一个文本或者警告文本可以拥有所需要尽可能多的文本段。

5.3.1 创建文本段

文本段在文本组或者报警组里管理。



按<Insert> 键将文本段添加到列表中。

Name	Type	StyleClass	Details
TextSnippet_1	IndexText	default	Text(<none>, <none>, ...)

文本段在属性中配置。

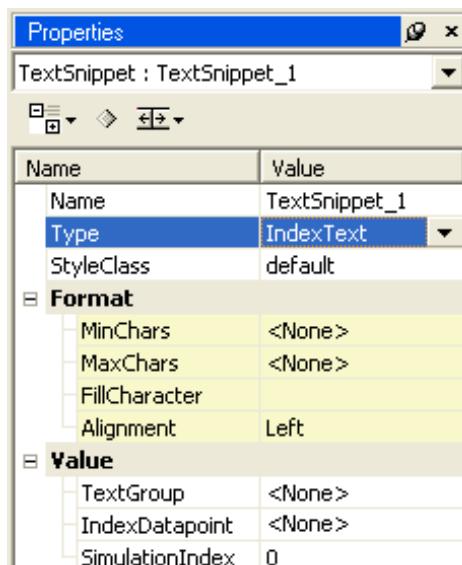


图22 文本段属性

5.3.2 文本段类型

可以使用以下文本段：

类型	描述	使用范围
Index Text	从另一个文本组中显示一个文本。运行时通过索引数据点选择文本。	文本组
DateTime	通过定义格式字符串显示日期和/或时间	文本组
Value	从数据点显示一个数字值（需要的话也可以通过单位系统按比例调整）	文本组 报警组
String	从数据点显示一个文本	文本组 报警组

文本段的属性依赖于所用的类型。

<table border="1"> <tr> <td>Type</td><td>IndexText</td></tr> <tr> <td>StyleClass</td><td>default</td></tr> <tr> <td colspan="2">Format</td></tr> <tr> <td>MinChars</td><td><None></td></tr> <tr> <td>MaxChars</td><td><None></td></tr> <tr> <td>FillCharacter</td><td></td></tr> <tr> <td>Alignment</td><td>Left</td></tr> <tr> <td colspan="2">Value</td></tr> <tr> <td>TextGroup</td><td><None></td></tr> <tr> <td>IndexDatapoint</td><td><None></td></tr> </table>	Type	IndexText	StyleClass	default	Format		MinChars	<None>	MaxChars	<None>	FillCharacter		Alignment	Left	Value		TextGroup	<None>	IndexDatapoint	<None>	<table border="1"> <tr> <td>Type</td><td>String</td></tr> <tr> <td>StyleClass</td><td>default</td></tr> <tr> <td colspan="2">Format</td></tr> <tr> <td>Alignment</td><td>Left</td></tr> <tr> <td>MinChars</td><td><None></td></tr> <tr> <td>MaxChars</td><td><None></td></tr> <tr> <td>FillCharacter</td><td></td></tr> <tr> <td colspan="2">Value</td></tr> <tr> <td>Datapoint</td><td><None></td></tr> </table>	Type	String	StyleClass	default	Format		Alignment	Left	MinChars	<None>	MaxChars	<None>	FillCharacter		Value		Datapoint	<None>						
Type	IndexText																																												
StyleClass	default																																												
Format																																													
MinChars	<None>																																												
MaxChars	<None>																																												
FillCharacter																																													
Alignment	Left																																												
Value																																													
TextGroup	<None>																																												
IndexDatapoint	<None>																																												
Type	String																																												
StyleClass	default																																												
Format																																													
Alignment	Left																																												
MinChars	<None>																																												
MaxChars	<None>																																												
FillCharacter																																													
Value																																													
Datapoint	<None>																																												
<table border="1"> <tr> <td>Type</td> <td>Numeric</td> </tr> <tr> <td>StyleClass</td> <td>default</td> </tr> <tr> <td colspan="2">Format</td> </tr> <tr> <td>AddFractionDigits</td> <td>0</td> </tr> <tr> <td>Alignment</td> <td>Right</td> </tr> <tr> <td>MinChars</td> <td><None></td> </tr> <tr> <td>MaxChars</td> <td><None></td> </tr> <tr> <td>FillCharacter</td> <td></td> </tr> <tr> <td>Content</td> <td>Value Unittext</td> </tr> <tr> <td>MinIntegerDigits</td> <td>1</td> </tr> <tr> <td>UnitText</td> <td>Abbreviation</td> </tr> <tr> <td colspan="2">Value</td> </tr> <tr> <td>Datapoint</td> <td><None></td> </tr> </table>	Type	Numeric	StyleClass	default	Format		AddFractionDigits	0	Alignment	Right	MinChars	<None>	MaxChars	<None>	FillCharacter		Content	Value Unittext	MinIntegerDigits	1	UnitText	Abbreviation	Value		Datapoint	<None>	<table border="1"> <tr> <td>Type</td> <td>DateTime</td> </tr> <tr> <td>StyleClass</td> <td>default</td> </tr> <tr> <td>SimulationValue</td> <td>0</td> </tr> <tr> <td colspan="2">Format</td> </tr> <tr> <td>MinChars</td> <td><None></td> </tr> <tr> <td>MaxChars</td> <td><None></td> </tr> <tr> <td>FillCharacter</td> <td></td> </tr> <tr> <td>Alignment</td> <td>Center</td> </tr> <tr> <td>TextGroup</td> <td><None></td> </tr> </table>	Type	DateTime	StyleClass	default	SimulationValue	0	Format		MinChars	<None>	MaxChars	<None>	FillCharacter		Alignment	Center	TextGroup	<None>
Type	Numeric																																												
StyleClass	default																																												
Format																																													
AddFractionDigits	0																																												
Alignment	Right																																												
MinChars	<None>																																												
MaxChars	<None>																																												
FillCharacter																																													
Content	Value Unittext																																												
MinIntegerDigits	1																																												
UnitText	Abbreviation																																												
Value																																													
Datapoint	<None>																																												
Type	DateTime																																												
StyleClass	default																																												
SimulationValue	0																																												
Format																																													
MinChars	<None>																																												
MaxChars	<None>																																												
FillCharacter																																													
Alignment	Center																																												
TextGroup	<None>																																												

5.3.3 在文本中集成文本段

从文本组或者报警组中将文本段添加到文本中。

下面例子中，当前页码以两位十进制数输出到页面描述中。

Name	Type	StyleClass	Details
{12} sfpicNumber	Numeric	default	Datapoint(PicNumber)

注意

文本必须设置为编辑模式从而将文本段与文本相连。我们建议在文本列表而不是文本属性中输入文本。

通过单击右键并选择文本段将文本插入所需的位置。

Index	English	German
0	Main Menue	Hauptmenü
1	Display Text and Value	sfpicNumber Wert

文本段插入后，通过文本段名在{}中显示。

Index	English	German
0	Main Menue	{sfpicNumber}Hauptmenü

注意

文本段只能以一种语言添加到所选的文本中。

5.3.4 文本组和报警组中文本段的区别

当输出一个文本段时，在文本组和报警组的文本段中存在区别：

- 文本组中的文本

如果有变化，文本组中位于文本中的文本段内容会在运行时更新。

- 报警文本

当触发报警时，显示文本段的内容。有变化时报警文本不会更新。

5.4 输入锁定

与4.3部分中所述的密码控制一起，可以在用户层上锁定输入。作为选择，这个层的管理同样可以在应用程序中实现。

在"Runtime" 属性组中，可以为任何能够输入的控制配置锁定的类型。默认时锁定被禁止（即"Locking"属性设置为<Never>）。

根据锁定所需的类型，将"Locking Datapoint"中读取的值与定义的"Level"相比较，然后为输入锁定或者释放控制。

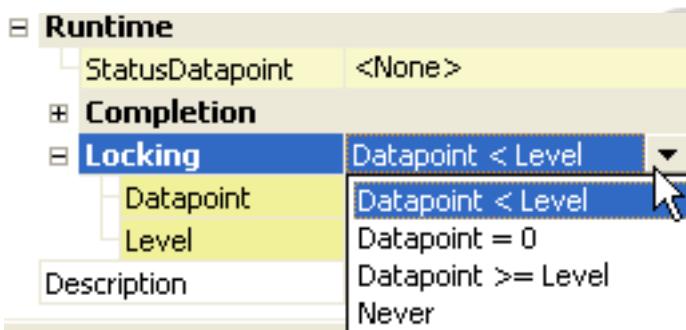


图23 输入锁定

内部数据源

6、内部数据源

内部数据通过内部数据源来管理。

Name	PLC-Type	VC-Type
Connection		
IPAddress	PLCSTRING	STRING
NodeNumber	USINT	INTEGER
DateTime		
Day	UINT	INTEGER
DayOfWeek	UINT	INTEGER
Hour	UINT	INTEGER
Minute	UINT	INTEGER
Month	UINT	INTEGER
Second	UINT	INTEGER
Year	UINT	INTEGER
Display		
Backlight	USINT	INTEGER
Brightness	USINT	INTEGER
Contrast	USINT	INTEGER
Memory		
LargestFreeBlock	UDINT	INTEGER
TotalFree	UDINT	INTEGER
TotalInstalled	UDINT	INTEGER
Temperature		
CPU	USINT	INTEGER
Room	USINT	INTEGER

图24 内部数据点

"Type" 属性决定了数据源是局部还是内部数据源。

按刷新按钮 或者<F5>键将内部数据点添加到数据源中。

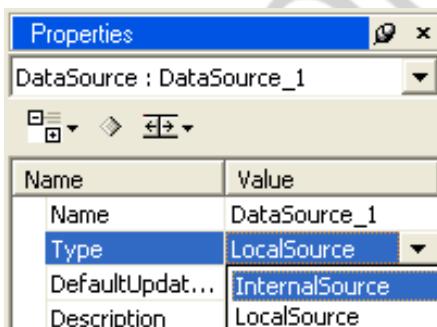


图25 内部数据源

重要

只有"Display" 和 "DateTime" 组中的数据点可以写。其他所有数据点必须是只读。

局部数据源用来连接用户任务和可视化程序的变量。

如果可视化程序中需要内部数据，那么必须按<Insert>键将内部数据添加到"Data sources"下的项目树中。

7、小结

没有什么不可能。在可视化组件编辑器和VISAPI库中函数的帮助下，可以实现可视化程序（即“操作和监控”）中的大部分命令。

这个模块描述了可视化组件编辑器的先进的功能和特殊属性以及VISAPI 库的函数。

通过对可视化组件编辑器基本元素的理解（培训模块TM610，TM640 和 TM650），及可视化组件帮助的协助，现在每个人都可以创建一个复杂的可视化应用程序来满足系统和操作者的要求。

可视化组件的范例也可以用来回答不同应用程序中可能仍然存在的问题。

可视化组件的范例也可以与可视化组件一起安装。项目安装在目录 <Automation Studio\Samples\VisualComponentsNG>下。范例项目的名字是 "Demo01.pgp"。

这个范例用德语、英语和汉语的形式提供了可视化组件编辑器中所有函数和控制的总结。在项目的用户任务中，将元素添加到各个屏幕页面中。

小结

Notes

ELECTRONIC DOCUMENT

Notes

ELECTRONIC DOCUMENT

小结

培训模块综述

- | | |
|-------------------------------------|-------------------------------|
| TM200 – 贝加莱B&R 公司介绍** | TM600 – 图文显示的基础 |
| TM201 – 贝加莱B&R 产品系列** | TM601 – 贝加莱人机界面产品** |
| TM210 – Automation Studio™ 基础 | TM610 – ASiV 的基础 |
| TM211 – Automation Studio™ 在线通信 | TM620 – ASiV 的维护* |
| TM212 – 自动化对象 (Target) ** | TM630 – 图文显示的编程规则 |
| TM213 – 自动化运行 (Runtime) 系统 | TM640 – ASiV 报警系统 |
| TM220 – 维护信息* | TM650 – ASiV 的国际化操作 |
| TM221 – 自动化组件和出错信息查询* | TM660 – ASiV 的远程操作 |
| TM223 – Automation Studio™ 诊断 | TM670 – ASiV 高级应用 |
| TM230 – 结构化软件编程 | TM700 – Automation Net PVI |
| TM231 – 面向机器设备的Automation Studio™ * | TM701 – PVI 通信* |
| TM240 – 梯形图(LAD) | TM710 – PVI DLL 编程 |
| TM241 – 功能块图 (FBD)* | TM711 – PVI 的服务 |
| TM242 – 连续功能图 (CFC)* | TM712 – PVIControl.NET |
| TM243 – 顺序功能图 (SFC)* | TM720 – PVI 维护和诊断* |
| TM245 – 指令表 (IL)* | TM730 – PVI OPC |
| TM246 – 结构文本 (ST) | TM800 – APROL 系统概念 |
| TM247 – Automation Basic (AB)* | TM801 – APROL 工程设计基础 |
| TM248 – ANSI C | TM810 – APROL 安装, 配置和恢复* |
| TM250 – 内存管理和数据存储 | TM811 – APROL 运行(Runtime) 系统* |
| TM260 – Automation Studio™ 函数库I | TM812 – APROL 操作员管理 |
| TM261 – Automation Studio™ 函数库 II* | TM813 – APROL XML 查询* |
| TM264 – 定时处理单元 (TPU) * | TM814 – APROL 审计追踪* |
| TM400 – 运动控制的基础 | TM820 – APROL 维护* |
| TM401 – 贝加莱B&R 运动控制产品** | TM830 – APROL 项目工程设计 |
| TM402 – 运动控制系统的计算* | TM840 – APROL 参数管理和配方 |
| TM410 – ASiM 的基础 | TM850 – APROL 控制器配置和INA 通讯 |
| TM440 – ASiM的基本功能 | TM860 – APROL 库设计 |
| TM441 – ASiM多轴运动功能 | TM861 – APROL 通讯互联* |
| TM445 – ACOPOS ACP10 软件 | TM865 – APROL 库指导手册 |
| TM446 – 电子凸轮* | TM870 – APROL Python 编程* |
| TM447 – ACOPOS 智能过程技术 (SPT) * | TM880 – APROL 报表* |
| TM450 – ACOPOS 控制理念和控制器设置 | ** 查看产品目录 |
| TM460 – 启动B&R 电机* | * 即将出版 |

全球总部

Bernecker+Rainer Industrie-Elektronik Ges.m.b.H.

B&R Straße 1

A-5142 Eggelsberg 奥地利

Tel.: +43(0)7748/6586-0

Fax: +43(0)7748/6586-26

info@br-automation.com

www.br-automation.com

中国总部

贝加莱工业自动化（上海）有限公司

上海市漕宝路70号光大会展中心C座16楼

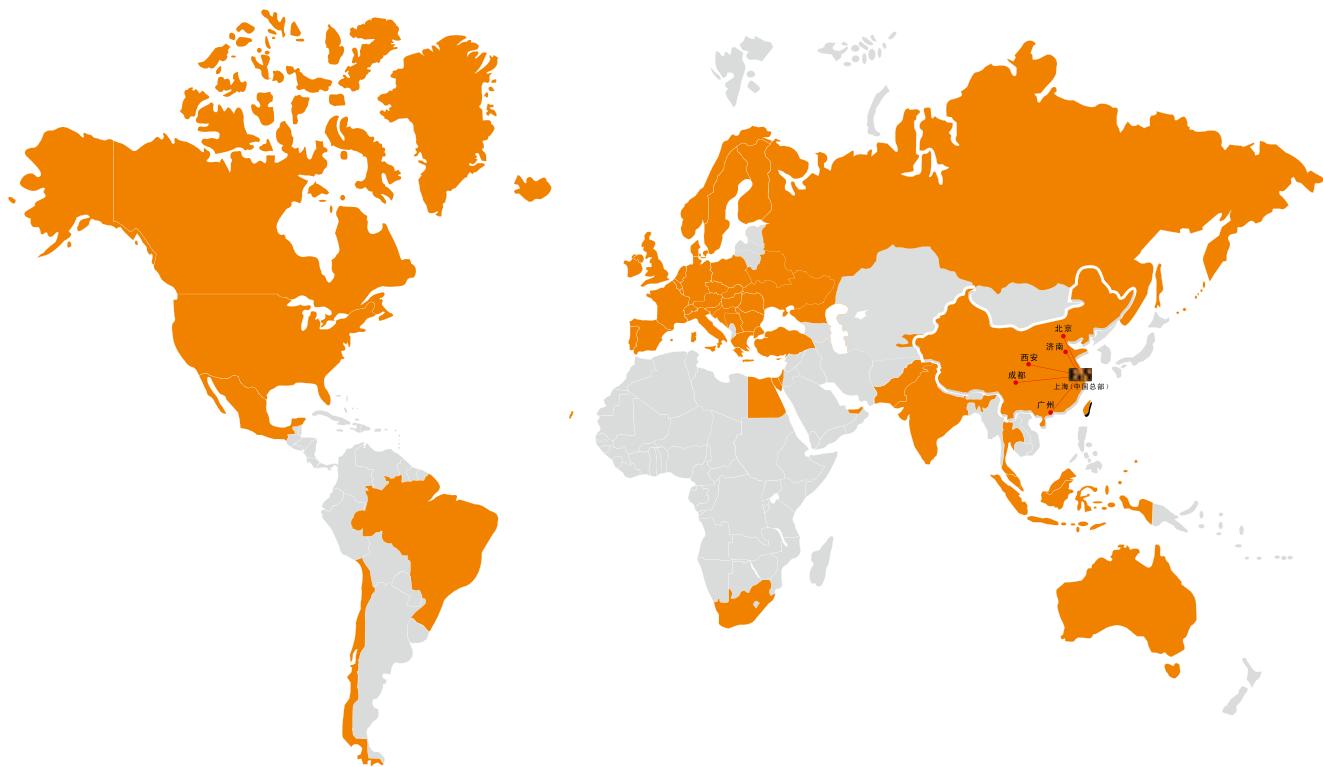
Tel.: +86/(0)21/6432 6000

Fax: +86/(0)21/6432 6108

info.cn@br-automation.com

www.br-automation.cn

全球50多个国家超过120个分支机构 www.br-automation.com/contact



中国总部



中国办事处

Austria · Australia · Belgium · Belarus · Brazil · Bulgaria · Canada · Chile · China · Croatia · Cyprus · Czech Republic · Denmark · Egypt · Emirates · Finland · France · Germany · Greece · Hungary · India · Indonesia · Ireland · Israel · Italy · Korea · Kyrgyzstan · Malaysia · Mexico · The Netherlands · Norway · Pakistan · Poland · Portugal · Romania · Russia · Singapore · Slovakia · Slovenia · South Africa · Spain · Sweden · Switzerland · Thailand · Turkey · Ukraine · United Kingdom · USA