

InTouch® HMI 脚本与逻辑指南

Invensys Systems, Inc.

修订版 A

最后修订日期：2007 年 8 月 6 日



版权声明

© 2007 Invensys Systems, Inc. 版权所有。保留所有权利。

保留所有权利。未经 Invensys Systems, Inc. 事先书面明确同意，不得通过任何手段（电子、机械、影印、录制或其它方式）复制、传输本文档中的任何部分，或是将其存储到检索系统。使用本文档所含信息不需承担任何相关的版权或专利责任。虽然在编制本文档的过程中已采取一切预防措施，但错误或疏漏在所难免，出版商与作者对此概不承担任何责任。对由于使用本文档所含信息而导致的任何损害，亦不承担任何赔偿责任。

本文档中的内容如有变更，恕不另行通知，这些内容亦不代表 Invensys Systems, Inc. 一方的承诺。本文所述软件系在遵守许可协议或保密协议的前提下提供。本软件的使用或复制必须遵守这些协议中的各项条款。

Invensys Systems, Inc.
26561 Rancho Parkway South
Lake Forest, CA 92630 U.S.A.
(949) 727-3200

<http://www.wonderware.com>

对产品文档如有任何意见或建议，请发送电子邮件到 productdocs@wonderware.com。

商标

本文所提及且已知为商标或服务标志的所有专用名词均已采用适当的首字母大写形式。Invensys Systems, Inc. 无法证实此类信息的准确性。在本文档中使用某个专用名词不应视为会影响任何商标或服务标志的有效性。

Alarm Logger、ActiveFactory、ArchestrA、Avantis、DBDump、DBLoad、DT Analyst、FactoryFocus、FactoryOffice、FactorySuite、FactorySuite A²、InBatch、InControl、IndustrialRAD、IndustrialSQL Server、InTouch、MaintenanceSuite、MuniSuite、QI Analyst、SCADAAlarm、SCADASuite、SuiteLink、SuiteVoyager、WindowMaker、WindowViewer、Wonderware 以及 Wonderware Logger 均为 Invensys plc 及其子公司与附属公司的商标。所有其它品牌可能是其相应所有者的商标。

目录

| | |
|-------------------------|----|
| 欢迎..... | 9 |
| 文档惯例..... | 9 |
| 技术支持..... | 10 |
| | |
| 第 1 章 脚本简介..... | 11 |
| 基本脚本概念 | 12 |
| 脚本类型..... | 12 |
| 编辑与创建脚本..... | 13 |
| 高级脚本概念 | 14 |
| OLE 对象..... | 14 |
| 使用 ActiveX 控件编写脚本 | 14 |
| | |
| 第 2 章 创建与编辑脚本 | 15 |
| 打开脚本进行编辑 | 16 |
| 保存或放弃对脚本的更改 | 17 |
| 复制、剪切及粘贴文本..... | 17 |
| 查找与 / 或替换文本 | 18 |
| 插入代码元素 | 18 |
| 访问脚本函数的帮助..... | 19 |
| 验证脚本的语法是否正确 | 19 |
| 打印脚本..... | 20 |
| 删除脚本..... | 20 |

| | |
|-------------------------------|---------------|
| 第 3 章 脚本触发器 | 21 |
| 脚本触发器的类型 | 22 |
| 使用多个触发器 | 22 |
| 定期执行脚本 | 22 |
| 配置应用程序脚本 | 23 |
| 应用程序脚本的限制 | 24 |
| 配置窗口脚本 | 24 |
| 配置键脚本 | 26 |
| 配置条件脚本 | 28 |
| 配置数据改变脚本 | 30 |
| 配置动作脚本 | 31 |
| 配置 ActiveX 事件脚本 | 34 |
| 在运行时暂停脚本执行 | 37 |
| \$LogicRunning 系统标记 | 37 |
| 第 4 章 脚本语言 | 39 |
| 基本语法规则 | 40 |
| 子程序 | 40 |
| 语句 | 40 |
| 缩进 | 40 |
| 注释 | 40 |
| 标记引用 | 41 |
| 数据值 | 41 |
| 值表达式 | 41 |
| 语法验证 | 41 |
| 调用标准函数 | 42 |
| 调用标准函数的语法 | 42 |
| 将参数传递给函数 | 43 |
| 调用自定义函数 (QuickFunction) | 44 |
| 将参数传递给 QuickFunction | 44 |
| 赋值语句与运算符 | 45 |
| 支持的运算符 | 45 |
| 设置运算符的求值顺序 | 51 |
| 隐式数据类型转换 | 52 |
| 表达式的示例 | 53 |
| 使用条件程序分支结构 | 54 |
| 简单条件结构 | 55 |

| | |
|------------------------------------|-----------|
| 嵌套条件结构 | 55 |
| 无效脚本示例（遗失 ENDIF） | 55 |
| 无效脚本示例（嵌套不正确） | 56 |
| 使用程序循环 | 56 |
| 强制结束循环 | 58 |
| 循环对其它运行时进程的影响 | 58 |
| 循环执行的时间限制 | 59 |
| 循环的示例 | 59 |
| 使用局部变量 | 60 |
| 声明局部变量 | 60 |
| 局部变量与标记之间的命名冲突 | 61 |
| | |
| 第 5 章 自定义脚本函数 | 63 |
| 关于 QuickFunction | 63 |
| 配置 QuickFunction | 64 |
| 调用 QuickFunction | 66 |
| 创建异步 QuickFunction | 66 |
| 异步 QuickFunction 的限制 | 66 |
| 检查是否有任何异步 QuickFunction 正在运行 | 67 |
| 停止运行异步 QuickFunction | 68 |
| | |
| 第 6 章 内置函数 | 69 |
| 在动画显示链接中强制更新 | 69 |
| 数学计算 | 70 |
| 舍入、截断及确定符号 | 70 |
| 使用三角函数 | 73 |
| 返回 Pi 的值 | 75 |
| 计算对数 | 76 |
| 计算平方根 | 77 |
| 字符串运算 | 78 |
| 返回字符串的某些部分 | 78 |
| 更改字符串大小写 | 80 |
| 从字符串中删除空格 | 81 |
| 使用空格设置字符串格式 | 82 |
| 在字符与 ASCII 码之间转换 | 82 |
| 搜索与替换字符串中的文本 | 83 |
| 返回关于字符串的信息 | 86 |
| 比较字符串 | 88 |

| | |
|-------------------------------|-----|
| 转换数据类型 | 90 |
| Text() 函数 | 91 |
| StringFromIntg() 函数 | 92 |
| StringFromReal() 函数 | 93 |
| StringToIntg() 函数 | 94 |
| StringToReal() 函数 | 95 |
| DText() 函数 | 96 |
| 在运行时处理 InTouch 窗口 | 97 |
| 显示打开的窗口的列表 | 97 |
| 检查窗口是打开、关闭或是否存在 | 97 |
| 打开 InTouch 窗口 | 98 |
| 移动窗口与调整大小 | 101 |
| 隐藏 InTouch 窗口 | 102 |
| 更改窗口的颜色 | 103 |
| 在运行时打印窗口 | 104 |
| 处理日期与时间信息 | 109 |
| 检索数值日期与时间信息 | 109 |
| 检索字符串日期与时间信息 | 114 |
| 将日期与时间信息转换为字符串 | 116 |
| 检查夏令时状态 | 119 |
| 与其它应用程序交互 | 120 |
| 启动 Windows 应用程序 | 120 |
| 检索正在运行的应用程序的应用程序标题 | 121 |
| 检查应用程序是否正在运行 | 121 |
| 激活正在运行的 Windows 应用程序 | 122 |
| 将模拟键击发送到应用程序 | 123 |
| 关闭、最小化或最大化 Windows 应用程序 | 125 |
| 使用 DDE 执行命令与交换数据 | 126 |
| 处理文件 | 129 |
| 管理文件 | 129 |
| 读取和写入 CSV 数据 | 133 |
| 读取和写入文本数据 | 136 |
| 检索系统相关信息 | 138 |
| 检索计算机的节点名 | 138 |
| 检索磁盘空间信息 | 139 |
| 检索文件或目录的有关信息 | 140 |
| 检索 Windows 环境有关的信息 | 141 |
| 检索 InTouch 相关信息 | 142 |

| | |
|----------------------------|-----|
| 检索 InTouch 应用程序目录的名称 | 142 |
| 检索 InTouch 版本 | 143 |
| 安全性相关脚本 | 144 |
| 登录与注销 | 144 |
| 更改与设置口令 | 144 |
| 指定与配置用户 | 145 |
| 管理安全性及其它信息 | 145 |
| 其它脚本 | 146 |
| 从应用程序 InTouch 播放声音文件 | 146 |
| 获取与设置向导属性 | 147 |

第 7 章 使用 OLE 对象编写脚本 153

| | |
|--|-----|
| 创建、验证及释放 OLE 对象 | 153 |
| OLE_CreateObject() 函数 | 154 |
| OLE_IsObjectValid() 函数 | 154 |
| OLE_ReleaseObject() 函数 | 155 |
| 使用 OLE 对象的属性与方法 | 156 |
| 访问 OLE 对象的属性 | 156 |
| 调用 OLE 对象的方法 | 157 |
| 将多个指针指定给相同的 OLE 对象 | 158 |
| 排解 OLE 错误 | 159 |
| OLE_GetLastObjectError() 函数 | 159 |
| OLE_GetLastObjectErrorMessage() 函数 | 159 |
| OLE_ResetObjectError() 函数 | 159 |
| OLE_ShowMessageOnObjectError() 函数 | 160 |
| OLE_IncrementOnObjectError() 函数 | 160 |
| 使用 OLE 的好处 | 161 |
| 随机号产生随机数 | 161 |
| 创建用户界面对话框 | 161 |
| 打开 Windows 日期与时间属性面板 | 163 |
| 读取和写入注册表 | 164 |
| 最小化窗口 | 164 |

第 8 章 编写 ActiveX 控件脚本 165

| | |
|-------------------------------------|-----|
| 调用 ActiveX 控件方法 | 165 |
| 从 InTouch HMI 访问 ActiveX 控件属性 | 167 |
| 配置 ActiveX 控件属性以读取和写入数据 | 167 |

| | |
|----------------------------------|-----|
| 创建与复用 ActiveX 事件脚本 | 169 |
| 创建 ActiveX 事件脚本 | 169 |
| 复用 ActiveX 事件脚本 | 171 |
| 创建自引用 ActiveX 事件脚本 | 172 |
| 导入 ActiveX 事件脚本 | 173 |
| 第 9 章 QuickScript 疑难排解 | 175 |
| 将消息记录到 Log Viewer..... | 175 |
| LogMessage() 函数 | 177 |
| 查看 Log Viewer 消息..... | 178 |
| 索引..... | 179 |

欢迎

您可以创建脚本，将操作程序添加到 InTouch 人机界面 (HMI) 应用程序。使用脚本可以将功能与特性添加到动画、报警管理、操作员界面以及趋势向导。

您可以联机查看本文，也可以使用 Adobe Acrobat Reader 的打印功能来打印本文的部分或全部内容。

在开始了解脚本与逻辑之前，必须知道如何使用 Microsoft Windows，包括浏览菜单、在应用程序之间切换，以及在屏幕上移动对象。如需有关这些任务的帮助，请参阅“Microsoft 帮助”。

文档惯例

本文采用以下惯例：

| 惯例 | 用于 |
|-----------|---------------------|
| 首字母大写 | 路径与文件名。 |
| 粗体 | 菜单、命令、对话框名称以及对话框选项。 |
| 等宽字体 | 代码范例与显示文本。 |

技术支持

Wonderware 的“技术支持”部门提供多种技术支持方案，帮助解答有关 Wonderware 产品及其实施方案的任何疑问。

在与“技术支持”部门联系之前，请参阅本文中相关的章节，以寻求问题的可能解决方案。如果需要联系技术支持以获取帮助，请准备好以下信息：

- 使用的操作系统的类型与版本。
- 有关如何重现问题的详细说明。
- 看到的错误消息的准确内容。
- **Log Viewer** 或任何其它诊断应用程序提供的任何相关输出列表。
- 为解决问题所作的尝试及其结果的详细说明。
- 如果遇到仍然存在的已知问题，请提供指定给该问题的“Wonderware 技术支持”案例号。

第 1 章

脚本简介

您可以使用 InTouch 脚本语言 QuickScript 构建更强大可靠的应用程序。程序提供七种类型的脚本和许多内置的脚本函数。

这七种类型的脚本都由导致执行它们的事件或条件来定义。例如，应用程序启动时、停止时或运行期间，执行应用程序脚本。数据的某一项发生更改时，执行数据改变脚本。窗口打开时、关闭时或保持打开期间，执行窗口脚本。

内置的脚本函数包括数学函数、三角函数、字符串函数等。使用这些函数可以帮助节省开发应用程序的时间。

InTouch 脚本可以包含“对象链接与嵌入”（Object Linking and Embedding，简称 OLE）对象与 ActiveX 控件。

您可以在脚本语言中使用条件语句、循环以及局部变量，以便在应用程序中创建复杂的效果。

基本脚本概念

在开始编写脚本之前，应该了解：

- **脚本**是指示应用程序执行相关任务的一组指令。
- **QuickScript** 是 InTouch HMI 脚本语言。
- **函数**是可以由另一个脚本调用的脚本。InTouch HMI 附带一组预定义的函数供您使用。
- **QuickFunction** 是在 QuickScript 中编写并在 QuickFunction 库中存储的可重复使用的函数。要创建 QuickFunction，只需创建一个 QuickScript 并给它命名即可。QuickFunction 可以由另一个脚本或是从动画链接表达式中调用。

脚本类型

在 InTouch 中，脚本按导致执行它的事件或条件进行分类。例如，如果希望在操作员按键盘上的某个键时执行脚本，则可以创建一个“键脚本”。

在选择脚本类型之后，便可以进一步定义导致脚本执行的准则或条件。例如，您可能希望在释放键而不是按下键的时候执行脚本。

脚本类型如下：

- **应用程序脚本**在 WindowViewer 运行时连续执行，或是在 WindowViewer 启动或关闭时执行一次。
- **窗口脚本**在 InTouch 窗口打开期间定期执行，或是在打开或关闭 InTouch 窗口时执行一次。
- **键脚本**在按下或释放特定的键或组合键时执行一次或定期执行。
- **条件脚本**在满足或不满足特定的条件时执行一次或定期执行。
- **数据改变脚本**在特定的标记或表达式的值发生改变时执行一次。
- **动作脚本**在操作员单击 InTouch HMI 图形对象时执行一次或定期执行。
- **ActiveX 事件脚本**在 ActiveX 事件（如单击 ActiveX 控件）发生时执行一次。

编辑与创建脚本

使用 “InTouch HMI 脚本编辑器” 可以在 InTouch WindowMaker 中创建并编辑脚本。



本例是应用程序脚本。每种类型的脚本都有自己的脚本对话框，其中包含该脚本类型所特有的选项与选择。

编辑器的标题栏指出正在处理的脚本类型。如需有关脚本类型的详细信息，请参阅第 12 页的“脚本类型”。

在 QuickScript 编辑器的底部。有一些文本、逻辑以及数学运算符按钮，您可以单击它们，以便在脚本中的光标位置插入该关键字、函数或符号。

条件框包含可供正在编写的脚本类型使用的执行条件。

仅当“制造工程模块”（Manufacturing Engineering Module，简称 MEM）随 InTouch HMI 程序一起安装时，右下角才会出现 **MEM OLE** 按钮。通过单击此按钮，可以使用 MEM 编写脚本。

高级脚本概念

一些高级脚本功能可以实现那些基本 InTouch HMI 之外的复杂功能。

通过使用 OLE 对象与 ActiveX 控件，可以访问本地计算机系统函数，还能与其它程序（如“制造工程模块”）进行交互。

OLE 对象

在自定义的脚本中，可以调用 OLE 对象。OLE 对象允许访问本地计算机系统函数，或是与其它程序（如“制造工程模块”）进行交互。

例如，使用 OLE 可以：

- 产生随机数。
- 创建用户界面对话框。
- 打开 Windows 日期与时间属性面板。
- 读写注册表。
- 最小化窗口。

使用 ActiveX 控件编写脚本

InTouch HMI 的“向导”菜单中提供多个 ActiveX 控件。由于 InTouch HMI 基于 Windows 操作环境，因此它几乎可以使用任何 ActiveX 控件。

第 2 章

创建与编辑脚本

创建新脚本的步骤根据脚本类型的不同而各异。一般而言，需要打开脚本编辑器，选择条件类型，输入语句，然后保存脚本。

如需有关创建每种类型的脚本的详细信息，请参阅下面几节：

- 第 23 页的“配置应用程序脚本”。
- 第 24 页的“配置窗口脚本”。
- 第 26 页的“配置键脚本”。
- 第 28 页的“配置条件脚本”。
- 第 30 页的“配置数据改变脚本”。
- 第 31 页的“配置动作脚本”。
- 第 34 页的“配置 ActiveX 事件脚本”。

如需了解基本的编辑操作以及一些可以节省时间的高级功能，请参阅下面几节。

- 第 16 页的“打开脚本进行编辑”。
- 第 17 页的“保存或放弃对脚本的更改”。
- 第 17 页的“复制、剪切及粘贴文本”。
- 第 18 页的“查找与 / 或替换文本”。
- 第 18 页的“插入代码元素”。
- 第 19 页的“访问脚本函数的帮助”。

打开脚本进行编辑

打开现有脚本的步骤根据脚本类型的不同而略有不同。

要打开应用程序脚本

- 1 执行以下操作之一：
 - 使用**经典视图**，在**脚本**窗格中双击**应用程序**。
 - 在**特别菜单**上，指向**脚本**，然后单击**应用程序脚本**。
- 2 在**条件类型**列表中，单击要编辑的脚本类型。

要打开窗口脚本

- 1 执行以下任何操作：
 - 使用**经典视图**，在**窗口**窗格中，使用鼠标右键单击窗口名，然后单击**窗口脚本**。
 - 使用**项目视图**展开**脚本**，然后双击该脚本。
 - 打开脚本与之关联的窗口。在**特别菜单**上，指向**脚本**，然后单击**窗口脚本**。
 - 打开脚本与之关联的窗口。使用鼠标右键单击窗口中的空白区域，然后单击**窗口脚本**。
- 2 在**条件类型**列表中，单击导致脚本运行的条件。

要打开 ActiveX 事件脚本

- ◆ 执行以下任何操作：
 - 使用**经典视图**，在**脚本**窗格中，展开 **ActiveX 事件**，然后双击脚本名。
 - 使用**项目视图**展开**脚本**，然后双击该脚本。
 - 双击脚本与之关联的 **ActiveX 控件**实例。单击**事件**选项卡，然后双击包含脚本名的单元。

要打开动作脚本

- 1 打开包含动作脚本与之关联的图形元素的窗口。
- 2 双击动作脚本与之关联的图形元素。
- 3 在**触动按钮**区域，单击**动作**。此时出现“脚本编辑器”。
- 4 在**条件类型**列表中，单击导致脚本运行的动作。

要打开键、条件或数据改变脚本

- 1 执行以下任何操作：
 - 使用**经典视图**，在**脚本**窗格中，展开脚本类别，然后双击脚本名。
 - 使用**项目视图**展开**脚本**，然后双击该脚本。
 - 在**特别**菜单上，指向**脚本**，然后单击相关的脚本类型。此时出现“脚本编辑器”。单击**浏览**按钮，然后单击脚本名。
- 2 如果适用，在**条件类型**列表中，单击导致脚本运行的条件。

保存或放弃对脚本的更改

在“脚本编辑器”中工作或完成时，可以手工或自动保存脚本。或者也可以完全放弃它。

恢复选项在窗口与应用程序脚本中不可用。

备注 保存或放弃更改总是适用于一类脚本的所有条件类型，而不只是当前可见的条件类型。

要保存更改并让脚本保持打开

- ◆ 在**脚本**菜单上，单击**保存**。

要保存更改并关闭脚本

- ◆ 单击**确定**。

要放弃更改并让脚本保持打开

- ◆ 单击**恢复**。

要放弃更改并关闭脚本

- ◆ 单击**取消**。

复制、剪切及粘贴文本

在“脚本编辑器”中，复制、剪切及粘贴文本与在任何其它 Windows 应用程序中的工作方式相同。使用标准的键盘快捷键 Ctrl-C、Ctrl-X 以及 Ctrl-V，或是工具栏按钮。

查找与 / 或替换文本

您可以在脚本中搜索并替换文本。

要查找与 / 或替换文本

- ◆ 在**编辑**菜单上，单击**查找**。此时出现**替换**对话框。此对话框中的选项与在其它 Windows 应用程序（如“记事本”）中的工作方式相同。

插入代码元素

通过从列表中进行选择，可以将各种代码元素自动插入脚本。这可以节省时间，并减小输入错误的风险。

要将函数插入脚本

- 1 在**插入**菜单上，指向**函数**，然后单击函数类别的名称。此时出现相应的**选择函数**对话框。
如果看不到感兴趣的函数，请单击列表底部的**下一页**以转到下一页。
- 2 单击要使用的函数。此时对话框关闭，该函数插入脚本中的光标位置。

要将标记名插入脚本

- 1 在**插入**菜单上，单击**标记名**。此时出现**选择标记**对话框。
- 2 单击要使用的标记名。
此外，也可以双击标记名以插入标记与点域列表中当前所选的点域。
- 3 要包含某个点域，请在**点域**列表中单击它。
- 4 单击**确定**。此时**选择标记**对话框关闭，该标记名（与点域，如果有）插入脚本中的光标位置。

如需有关使用**选择标记**对话框（包括设置多个标记源）的详细信息，请参阅 *InTouch® HMI 可视化指南* 中的第 4 章“设置对象动画效果”中的“选择 InTouch 标记”。

要将点域插入脚本

- 1 输入标记名与英文句点。
- 2 双击英文句点的右侧。此时出现**选择域名**对话框。
- 3 单击要使用的点域。此时对话框关闭，该点域插入脚本中的光标位置。

要将窗口名插入脚本

- 1 在**插入**菜单上，单击**窗口**。此时出现**要插入的窗口名**对话框。
- 2 单击要使用的窗口名。此时对话框关闭，窗口名插入脚本中的光标位置。

要将 ActiveX 方法或属性插入脚本

- 1 在**插入**菜单上，单击**ActiveX**。此时出现**ActiveX 控件浏览器**对话框。
- 2 在**控件名**列表中，单击希望列出其属性与方法的 ActiveX 控件实例。
- 3 在**方法 / 属性**列表中，单击方法或属性。
- 4 单击**确定**。此时对话框关闭，该方法或属性引用插入脚本中的光标位置。

要将关键字或运算符插入脚本

- ◆ 单击“脚本编辑器”底部的相关按钮。此时关键字或运算符插入脚本中的光标位置。

访问脚本函数的帮助

如需有关特定脚本函数的帮助，可以直接从“脚本编辑器”进行访问。

要查看关于特定脚本函数的帮助

- 1 在“脚本编辑器”的右下角，单击**帮助**。此时出现函数列表。
- 2 如果看不到感兴趣的函数，请单击列表底部的**下一页**以转到下一页。
- 3 单击所需函数的名称。此时出现相应的“帮助”主题。

验证脚本的语法是否正确

保存脚本时，“脚本编辑器”自动检查语法是否正确。如果发生错误，则出现一条包含更多信息的消息。您必须纠正所有的语法错误，才可以保存脚本。您还可以在编辑脚本的过程中手工开始验证。


要手工验证脚本语法

- ◆ 单击**验证**。

打印脚本

您可以从“脚本编辑器”中单独打印脚本，也可以使用 WindowMaker 中的打印功能打印特定类型的所有脚本。

要打印单个脚本

- 1 在“脚本编辑器”中打开脚本。请参阅第 16 页的“打开脚本进行编辑”。
-  2 单击工具栏中的**打印**按钮。此时脚本打印到 Windows 默认打印机。

要打印特定类型的所有脚本

- 1 在 WindowMaker 中的**文件**菜单上，单击**打印**。此时出现 **WindowMaker 打印件**对话框。
- 2 选择要打印的脚本类型对应的复选框。要打印所有文本，请单击**全部脚本**。
- 3 单击**下一步**。此时出现**选择输出目标**对话框。
- 4 执行以下操作之一
 - 单击**发送输出到打印机**。
 - 单击**发送输出到文本文件**。
- 5 单击**浏览**按钮选择打印机，或是查找文件。
- 6 单击**打印**。

删除脚本

删除脚本的步骤根据脚本类型的不同而各异。请参阅下面几节：

- 第 23 页的“配置应用程序脚本”。
- 第 24 页的“配置窗口脚本”。
- 第 26 页的“配置键脚本”。
- 第 28 页的“配置条件脚本”。
- 第 30 页的“配置数据改变脚本”。
- 第 31 页的“配置动作脚本”。
- 第 34 页的“配置 ActiveX 事件脚本”。

第 3 章

脚本触发器

所有的 InTouch HMI 脚本都由脚本触发器执行。每种类型的脚本都有一个或多个触发器用于启动它。

在“脚本编辑器”中，可以选择要用于执行脚本的脚本触发器。您可以根据执行脚本的时间与方式来选择脚本触发器。

您可以根据用户动作、内部状态与标记名值的改变来配置各种触发器。用户动作包括按键与单击图形元素。内部状态触发器可以包括启动 WindowViewer。

脚本由以下这些动作触发：

- 启动与关闭 WindowViewer。请参阅第 23 页的“配置应用程序脚本”。
- 打开与关闭窗口。请参阅第 24 页的“配置窗口脚本”。
- 按单键或组合键。请参阅第 26 页的“配置键脚本”。
- 满足特定条件，如标记名或表达式值。请参阅第 28 页的“配置条件脚本”。
- 更改标记名值或标记名字段值。请参阅第 30 页的“配置数据改变脚本”。
- 单击图形对象。请参阅第 31 页的“配置动作脚本”。
- 发生在 ActiveX 控件中的事件，如单击控件。请参阅第 34 页的“配置 ActiveX 事件脚本”。

此外，也可以暂停脚本执行。缺省条件下，启动 WindowViewer 时运行逻辑并执行脚本。在运行时，可以通过停止逻辑来暂停脚本执行。暂停之后可以恢复脚本执行。如需有关详细信息，请参阅第 37 页的“在运行时暂停脚本执行”。

脚本触发器的类型

在 InTouch HMI 中，脚本分为七种类型。每种类型的脚本都有一个或多个触发器可选择用于启动脚本。

- **应用程序脚本**有三个触发器：启动时、关闭时及运行期间。每个触发器都可以执行不同的脚本。
- **窗口脚本**有三个触发器：显示时、隐藏时及显示期间。每个触发器都可以执行不同的脚本。
- **键脚本**有三个触发器：放开时、按下时，或是按下期间。每个触发器都可以执行不同的脚本。
- **条件脚本**有四个触发器：为真时、为真期间、为假时以及为假期间。每个触发器都可以执行不同的脚本。
- 特定标记或表达式的值发生改变时，执行**数据改变脚本**。
- 操作员单击 InTouch HMI 图形对象时，**动作脚本**执行一次或定期执行。
- 发生特定的 ActiveX 事件（如单击 ActiveX 控件）时，**ActiveX 事件脚本**执行一次。

使用多个触发器

对于大多数脚本类型而言，都可以使用多个触发器，并可以将不同的脚本关联到每个触发器。

例如，您可以将一个应用程序脚本配置为在 WindowViewer 启动时执行某个脚本一次，在 WindowViewer 运行期间定期执行另一个脚本。

在**条件类型**列表中，选择触发器以查看触发器的现有脚本。

定期执行脚本

定期执行的脚本不是在触发之后立即执行，而是在指定的周期之后执行。

例如，如果将一个键脚本配置成在按下特定的键期间，每隔 5000 毫秒执行一次，则它会在按住该键 5 秒之后执行，此后每隔 5 秒执行一次。

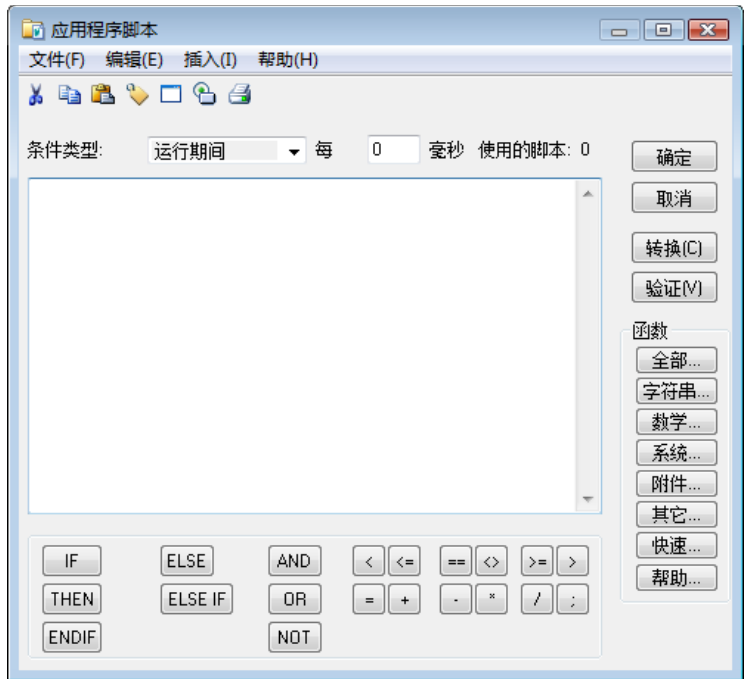
配置应用程序脚本

应用程序脚本链接到整个 InTouch HMI 应用程序。您可以使用应用程序脚本：

- 在 WindowViewer 启动时执行某个脚本一次。
- 在 WindowViewer 运行期间定期执行某个脚本。
- 在 WindowViewer 关闭时执行某个脚本一次。

要配置应用程序脚本

- 1 使用经典视图，在脚本窗格中，使用鼠标右键单击**应用程序**，然后单击**打开**。此时出现**应用程序脚本**对话框。



- 2 在**条件类型**列表中，单击脚本执行的条件：
 - 单击**启动时**，以便将脚本配置成在 WindowViewer 启动时执行一次。
 - 单击**运行期间**，以便将脚本配置成在 WindowViewer 运行期间定期执行。
 - 单击**关闭时**，以便将脚本配置成在 WindowViewer 关闭时执行一次。
- 3 如果在上一步中选择了**运行期间**，请在**每**框中输入 1 到 360000 毫秒之间的一个时间间隔。该时间间隔指定脚本执行的频率。
- 4 在窗口中输入脚本。
- 5 单击**确定**。

要删除应用程序脚本

- 1 使用**经典视图**，在**脚本**窗格中，使用鼠标右键单击**应用程序**，然后单击**打开**。此时出现**应用程序脚本**对话框。
- 2 在**条件类型**列表中，单击要删除的脚本的条件。此时脚本出现在**应用程序脚本**对话框的主要部分。
- 3 在**编辑**菜单上，单击**清除**。此时该脚本从主要部分中清除，关联的脚本也被删除掉。

应用程序脚本的限制

在启动或关闭 WindowViewer 时执行的应用程序脚本在同其它对象进行交互方面存在一定的限制。

您无法使用“启动时”应用程序脚本：

- 引用 ActiveX 方法、属性或事件。
- 对控件与 I/O 标记名或远程引用进行读取或写入。
- 运行数据改变脚本与条件脚本。

您无法使用“关闭时”应用程序脚本：

- 对控件与 I/O 标记名或远程引用进行读取或写入。
- 启动其它应用程序。

配置窗口脚本

窗口脚本是链接到特定窗口的脚本。您可以使用窗口脚本：

- 在打开 InTouch 窗口时执行某个脚本一次。
- 在打开 InTouch 窗口期间定期执行某个脚本。
- 在关闭 InTouch 窗口时执行某个脚本一次。

备注 打开 InTouch 窗口也称为“显示 InTouch 窗口”。关闭 InTouch 窗口也称为“隐藏 InTouch 窗口”。

要配置窗口脚本

- 1 使用经典视图，在窗口窗格中，使用鼠标右键单击窗口，然后单击窗口脚本。此时出现窗口脚本 - 窗口名 对话框。



- 2 在条件类型列表中，执行以下操作之一：
 - 单击显示时，以便将脚本配置成在关联的窗口启动时执行一次。
 - 单击显示期间，以便将脚本配置成在关联的窗口打开期间定期执行。
 - 单击隐藏时，以便将脚本配置成关联的窗口关闭时执行一次。
- 3 如果在上一步中选择显示期间，请在每框中输入介于 1 到 360000 毫秒之间的时间间隔。
- 4 在窗口中输入脚本。
- 5 单击确定。

要删除窗口脚本

- 1 使用经典视图，在窗口窗格中，使用鼠标右键单击窗口，然后单击窗口脚本。此时出现窗口脚本 - 窗口名 对话框。
- 2 在条件类型列表中，单击要删除的脚本的脚本触发器。此时脚本出现在窗口脚本 - 窗口名 对话框的主要部分。
- 3 在编辑菜单上，单击清除。

配置键脚本

键脚本是链接到特定单键或组合键的按键动作的脚本。您可以使用键脚本：

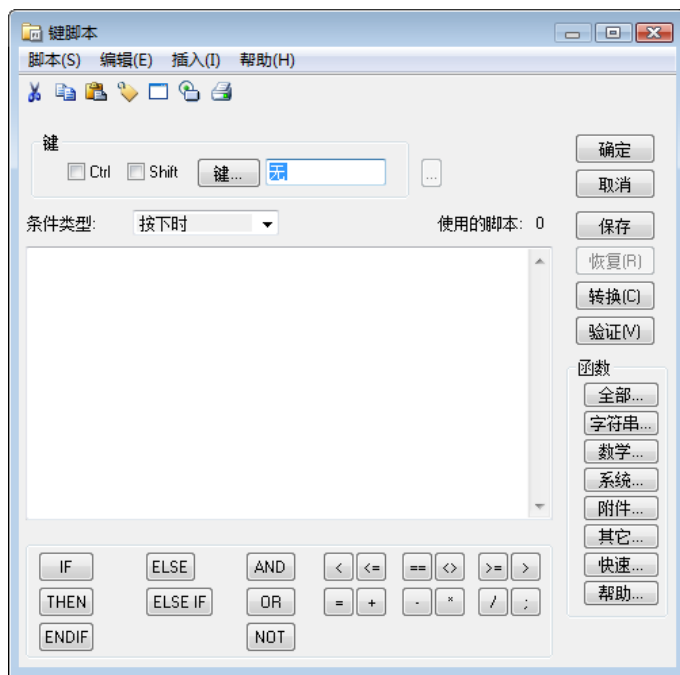
- 在按单键或组合键时执行某个脚本一次。
- 在按住或未按某个单键或组合键时定期执行某个脚本。
- 在释放单键或组合键时执行某个脚本一次。

键脚本由启动脚本的键的名称来确定。例如：**Ctrl+Q**。

备注 如果已经将一个动作脚本配置成使用相同的单键或组合键来触发，则忽略键脚本而执行动作脚本。

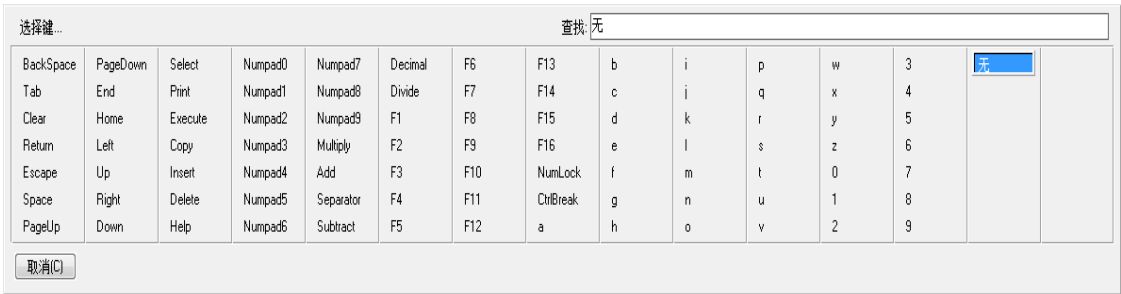
要配置键脚本

- 1 使用**经典视图**，在**脚本**窗格中，执行以下操作之一：
 - 要配置新的键脚本，使用鼠标右键单击**键**，然后单击**新建**。此时出现**键脚本**对话框。



- 要配置现有的键脚本，展开**键**，使用鼠标右键单击脚本名，然后单击**编辑**。此时出现**编辑键脚本**对话框。

2 单击**键**并从**选择键**对话框中选择一个键。



- 3 选择 **Ctrl** 与 / 或 **Shift** 复选框，以便指定 **Ctrl** 与 / 或 **Shift** 键同所选键的组合。
- 4 在**条件类型**列表中，执行以下操作之一：
- 单击**按下时**，以便将脚本配置成在按关联的单键或组合键时执行一次。
 - 单击**按下期间**，以便将脚本配置成在按住关联的单键或组合键期间定期执行。
 - 单击**放开时**，以便将脚本配置成在释放关联的单键或组合键时执行一次。
- 5 如果在上一步中选择**按下期间**，请在**每框**中输入介于 1 到 360000 毫秒之间的时间间隔。
- 6 在窗口中输入脚本。
- 7 单击**确定**。

要删除与某个键关联的所有键脚本

- ◆ 使用**经典视图**，在**脚本**窗格中，展开**键**，使用鼠标右键单击键脚本名，然后单击**删除**。出现消息时，单击**是**。

要删除与某个键关联的某个键脚本

- 1 使用**经典视图**，在**脚本**窗格中，展开**键**，使用鼠标右键单击键脚本名，然后单击**编辑**。此时出现**编辑键脚本**对话框。
- 2 在**条件类型**列表中，单击要删除的脚本的脚本触发器。此时脚本出现在**编辑键脚本**对话框的主要部分。
- 3 在**编辑**菜单上，单击**清除**。此时该脚本从主要部分中清除，关联的脚本也被删除掉。

配置条件脚本

条件脚本根据何时满足特定的逻辑条件而触发。使用条件脚本：

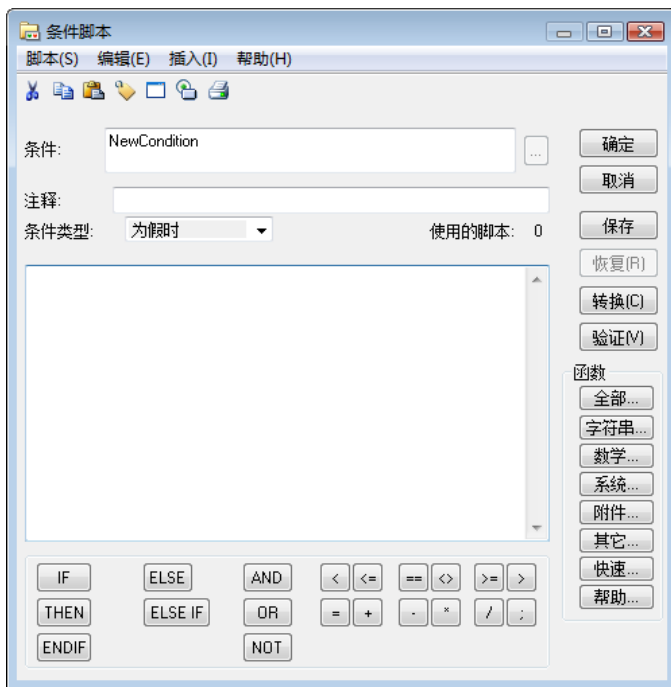
- 在满足条件时执行某个脚本一次。
- 在不满足条件时执行某个脚本一次。
- 在满足特定的条件期间定期执行某个脚本。
- 在不满足特定的条件期间定期执行某个脚本。

条件脚本由启动脚本的条件语法来确定。例如：`tag1>=13`。

备注 指定了“为真时”条件类型的脚本仅当条件从“假”转变为“真”时执行。指定了“为假时”条件类型的脚本仅当条件从“真”转变为“假”时执行。

要配置条件脚本

- 1 使用**经典视图**，在**脚本**窗格中执行以下操作之一：
 - 使用鼠标右键单击**条件**，然后单击**新建**。此时出现**条件脚本**对话框。



- 要编辑现有的条件脚本，请单击**条件**旁边的加号，使用鼠标右键单击条件脚本名，然后单击**编辑**。此时出现**编辑条件脚本**对话框。

- 2 在**条件**框中，输入要用作条件的表达式。
- 3 您可以在**注释**框中输入注释。
- 4 在**条件类型**列表中，执行以下操作之一：
 - 单击**为假时**，以便将脚本配置成在条件变为假时执行一次。
 - 单击**为假期间**，以便将脚本配置成在条件为假期间定期执行。
 - 单击**为真时**，以便将脚本配置成在条件变为真时执行一次。
 - 单击**为真期间**，以便将脚本配置成在条件为真期间定期执行。
- 5 如果在上一步中选择**为假期间**或**为真期间**，请在**每**框中输入介于 1 到 360000 毫秒之间的时间间隔。
- 6 输入脚本，或修改窗口中现有的脚本。
- 7 单击**确定**。

要删除与某个条件关联的所有条件脚本

- ◆ 使用**经典视图**，在**脚本**窗格中，展开**条件**，使用鼠标右键单击条件脚本名，然后单击**删除**。出现消息时，单击**是**。

要删除与某个条件关联的单个条件脚本

- 1 使用**经典视图**，在**脚本**窗格中，展开**条件**，使用鼠标右键单击脚本名，然后单击**编辑**。此时出现**编辑条件脚本**对话框。
- 2 在**条件类型**列表中，单击要删除的脚本的脚本触发器。此时脚本出现在**编辑条件脚本**对话框的主要部分。
- 3 在**编辑**菜单上，单击**清除**。此时该脚本从主要部分中清除，关联的脚本也被删除掉。

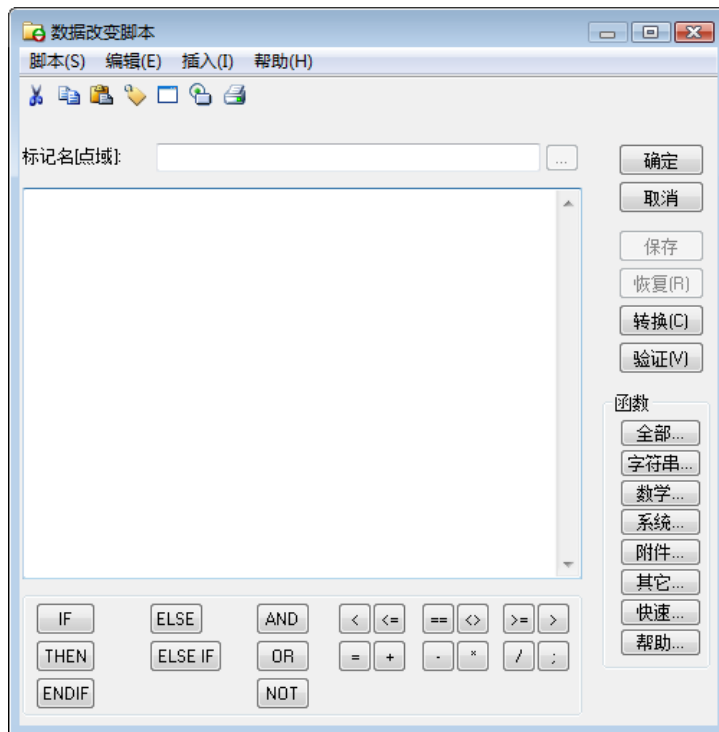
配置数据改变脚本

特定的标记名或点域的改变超过其定义的死区时，可以使用数据改变脚本执行某个脚本一次。

数据改变脚本由启动脚本的标记名或标记名字段来确定。例如：Tag1 或 Tag2.HiHiLimit。

要配置数据改变脚本

- 1 使用经典视图，在脚本窗格中，使用鼠标右键单击**数据改变**，然后单击**新建**。此时出现**数据改变脚本**对话框。



- 2 在**标记名 [点域]** 框中，输入标记名或标记名字段。
- 3 在窗口中输入脚本。
- 4 单击**确定**。

要删除数据改变脚本

- ◆ 使用经典视图，在脚本窗格中，展开**数据改变**，使用鼠标右键单击数据改变脚本名，然后单击**删除**。出现消息时，单击**是**。

配置动作脚本

使用动作脚本将操作员动作关联到图形对象。您可以使用图形对象配置一个或多个以下事件：

- 单击鼠标左键、中键或右键。
- 按住鼠标左键、中键或右键。
- 释放鼠标左键、中键或右键。
- 双击鼠标左键、中键或右键。
- 按单键或组合键。
- 按住单键或组合键。
- 释放单键或组合键。
- 将鼠标指针移到某个对象上。

动作脚本只能在对象自身的**动画链接选择**面板中配置。

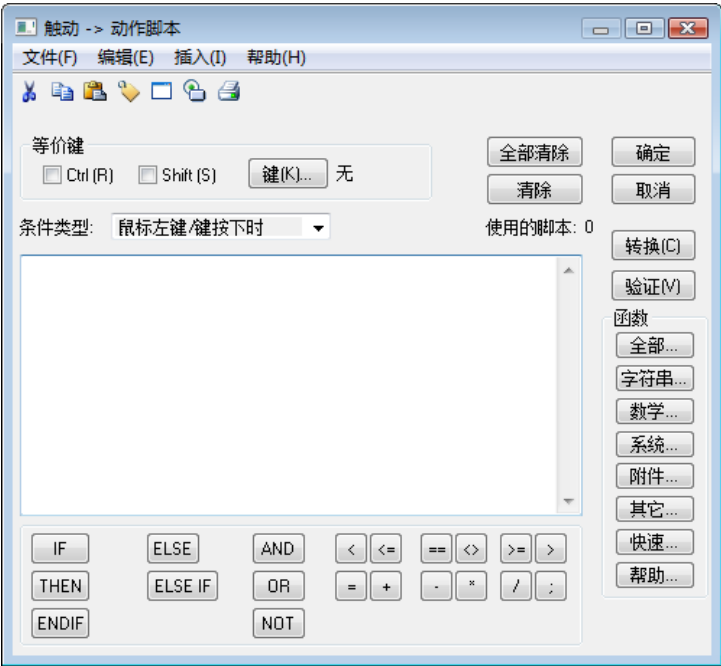
重要 如果存在与动作脚本相同的单键或组合键来触发的键脚本，则执行动作脚本并忽略键脚本。

要配置动作脚本

- 1 双击图形对象。此时出现**动画链接选择**面板。



2 单击**动作**。此时出现**触动 -> 动作脚本**对话框。



3 在**条件类型**列表中，单击以下对象之一：

| 要将某个脚本配置成在此条件下执行 | 单击 |
|------------------------|--------------|
| 按鼠标左键、特定的单键或组合键时执行一次 | 鼠标左键 / 键按下时 |
| 按住鼠标左键、特定的单键或组合键期间定期执行 | 鼠标左键 / 键按下期间 |
| 释放鼠标左键、特定的单键或组合键时执行一次 | 鼠标左键 / 键放开时 |
| 双击鼠标左键时执行一次 | 双击鼠标左键时 |
| 按鼠标右键时执行一次 | 单击鼠标右键时 |
| 按住鼠标右键期间定期执行 | 鼠标右键按下期间 |
| 释放鼠标右键时执行一次 | 鼠标右键放开时 |
| 双击鼠标右键时执行一次 | 双击鼠标右键时 |
| 按鼠标中键时执行一次 | 鼠标中键单击时 |
| 按住鼠标中键期间定期执行 | 鼠标中键按下期间 |
| 释放鼠标中键时执行一次 | 鼠标中键放开时 |
| 双击鼠标中键时执行一次 | 鼠标中键双击时 |

要删除单个动作脚本

- 1 双击包含要删除的动作脚本的图形对象。此时出现对象属性面板。
- 2 单击**动作**按钮。此时出现**触动 -> 动作脚本**对话框。
- 3 在**条件类型**列表中，单击脚本触发器。
- 4 在**编辑**菜单上，单击**清除**。此时该脚本从主要部分中清除，关联的脚本也被删除掉。

配置 ActiveX 事件脚本

使用 ActiveX 事件脚本在发生 ActiveX 事件时运行某个脚本。根据具体的 ActiveX 控件，这些事件可以包括：

- 启动 ActiveX 控件：**启动**
- 关闭 ActiveX 控件：**关闭**
- 用户单击 ActiveX 控件：**单击**
- 用户双击 ActiveX 控件：**双击**

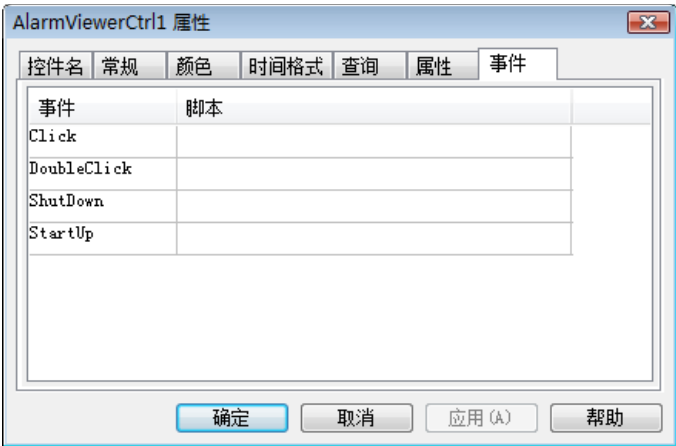
ActiveX 事件脚本通过名称来确定。缺省条件下，InTouch HMI 自动添加脚本与之关联的控件名与事件。例如：
MyActiveXScript (AlarmViewerCtrl1::Click)。

要配置新的 ActiveX 事件脚本

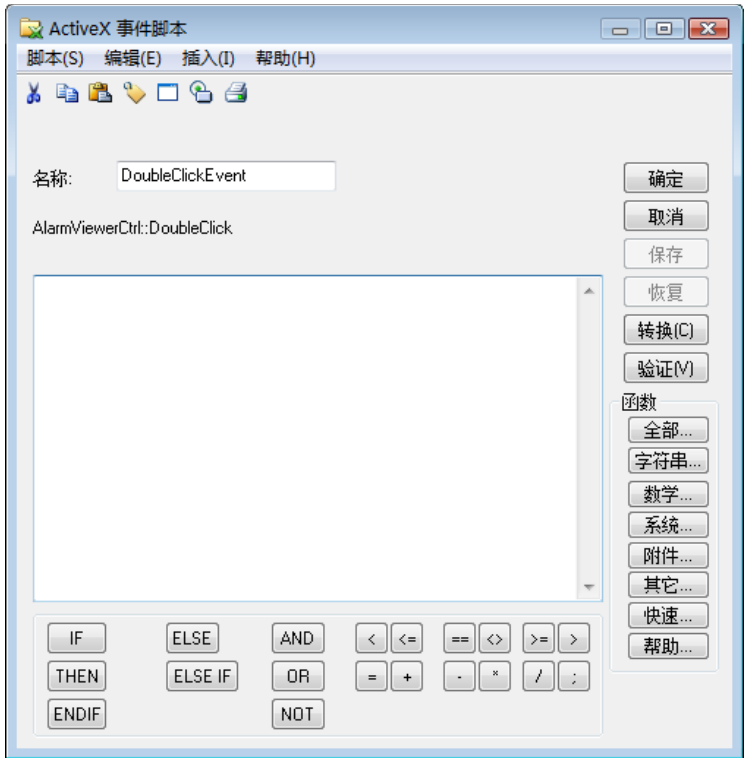
- 1 双击要配置的 ActiveX 控件。此时出现该 ActiveX 控件的属性对话框。



2 单击**事件**选项卡。



- 3 选择某个事件，如 Click（单击）、DoubleClick（双击）、ShutDown（关闭），或 StartUp（启动）。
- 4 单击该事件的**脚本**单元。此时出现方括号。
- 5 为事件脚本输入新的名称，然后单击**确定**。出现消息时，单击**确定**以创建一个新脚本。此时出现 **ActiveX 事件脚本**对话框。



- 6 在**名称**框中，可以更改 ActiveX 事件脚本名。
- 7 在窗口中输入脚本。
- 8 单击**确定**。

要编辑现有的 ActiveX 事件脚本

- 1 使用**经典视图**，在**脚本**窗格中，展开 **ActiveX 事件**，使用鼠标右键单击 ActiveX 脚本名，然后单击**编辑**。此时出现 **ActiveX 事件脚本**对话框。
- 2 对脚本进行必要的更改，然后单击**确定**。

要删除现有的 ActiveX 事件脚本

- 1 确保没有 ActiveX 控件正在使用要删除的 ActiveX 事件脚本。如果有 ActiveX 控件正在使用该脚本，请先执行以下操作：
 - a 从可能在使用 ActiveX 事件脚本的每个 ActiveX 控件的**事件面板**中，删除该 ActiveX 事件脚本引用。
 - b 关闭所有窗口并更新使用计数。
- 2 使用**经典视图**，在**脚本**窗格中，展开 **ActiveX 事件**，使用鼠标右键单击 ActiveX 脚本名，然后单击**删除**。出现消息时，单击**是**。此时该 ActiveX 事件脚本已删除。

在运行时暂停脚本执行

缺省条件下，启动 WindowViewer 时运行逻辑并执行同步脚本。在运行时，可以通过停止逻辑来暂停脚本执行。暂停之后可以恢复脚本执行。

要在运行时从菜单中暂停脚本执行

- ◆ 在**逻辑**菜单上，单击**停止逻辑**。此时同步脚本停止运行。异步脚本继续运行，但不会启动新的异步脚本。

要在运行时使用脚本暂停脚本执行

- ◆ 将值 0 写入离散系统标记 **\$LogicRunning**。此时同步脚本停止运行。异步脚本继续运行，但不会启动新的异步脚本。

要在运行时恢复脚本执行

- ◆ 在**逻辑**菜单上，单击**开始逻辑**。此时脚本恢复执行。

要在运行时使用脚本恢复脚本执行

- ◆ 将值 1 写入离散系统标记 **\$LogicRunning**。**\$LogicRunning** 系统标记必须包含在暂停逻辑时所执行的异步脚本中。

\$LogicRunning 系统标记

此系统标记监视与 / 或控制脚本的运行。

用法

\$LogicRunning

附注

将值设为 1 使脚本开始运行。将值设为 0 使脚本停止运行。

此系统标记相当于在 WindowViewer 的**逻辑**菜单上选择**开始逻辑**或**停止逻辑**。

您无法停止当前正在运行的异步脚本。不过可以阻止运行新的脚本。

数据类型

离散（可读写）

第 4 章

脚本语言

使用 InTouch HMI 脚本语言编写脚本时可以使用这些概念、技术及语法规则。

- 基本语法规则。请参阅第 40 页的“基本语法规则”。
- 调用预定义或自定义的函数。请参阅第 42 页的“调用标准函数”与第 44 页的“调用自定义函数 (QuickFunction)”。
- 使用赋值语句与各种运算符。请参阅第 45 页的“赋值语句与运算符”。
- 使用条件语句。请参阅第 54 页的“使用条件程序分支结构”。
- 使用循环。请参阅第 56 页的“使用程序循环”。
- 使用局部变量。请参阅第 60 页的“使用局部变量”。

如需有关脚本编辑器常规操作的详细信息，请参阅第 2 章“创建与编辑脚本”。

如需有关各种类型脚本触发器的详细信息，请参阅第 3 章“脚本触发器”。

如需标准脚本函数的参考，请参阅第 6 章“内置函数”。

基本语法规则

基本语法规则涉及 InTouch HMI 脚本语言的以下这些概念：

- 子程序
- 语句
- 缩进
- 注释
- 标记引用
- 数据值
- 值表达式
- 语法验证

子程序

在相同的脚本中没有单独子程序的概念，如 Visual Basic 中的“子”过程。要使用多个子程序构建一个脚本，必须为每个子程序都创建一个自定义的 QuickFunction。请参阅第 5 章“自定义脚本函数”。

语句

- 语句可以是赋值、函数调用或控制结构。
- 脚本中的每个语句都必须以英文分号 (;) 结尾。
- 只要每个语句都以英文分号结尾，同一行中便可以有多个语句。
- 通过使用换行符（按 Enter 键），可以将一条语句分布到多个行上。

缩进

您可以按任何方式缩进脚本代码。缩进没有功能相关性。

注释

要将文本标记为注释，请使用大括号 { } 将它括起来。注释可以跨越多行。

标记引用

有多种方式进行标记引用。

- 要引用本地“标记名字典”中定义的标记，只需简单地使用标记名即可。
- 要引用特定的点域，请使用常规引用格式 (Tagname.Dotfield)。
- 要引用远程节点上的数据项，请使用常规远程标记引用 (AccessName:Item)。
- 您还可以定义局部变量，它们的作用范围限制为当前脚本。请参阅第 60 页的“使用局部变量”。

数据值

- 您可以使用十进制或十六进制格式来指定整数值。例如，255 或 0xFF。
- 您可以使用十进制或科学计数法来指定浮点值。例如，0.001 或 1E-3。
- 要指定布尔值，请将数值 0 用作 FALSE、1 用作 TRUE。
- 要指定字符串值，请使用英文双引号将它括起来。例如：
"This is a string."

值表达式

值表达式可以包含值、标记引用以及函数调用，所有这些都通过适当的运算符链接起来。请参阅第 45 页的“赋值语句与运算符”。

语法验证

保存脚本时，“脚本编辑器”自动检查语法是否正确。通过单击 **验证** 按钮，也可以手工开始这项验证。请参阅第 19 页的“验证脚本的语法是否正确”。

调用标准函数

标准函数是 InTouch HMI 中预定义好的。请参阅第 44 页的“调用自定义函数 (QuickFunction)”。

调用标准函数的语法

调用预定义脚本函数的语法取决于函数是否以及如何返回结果。

有些函数不返回任何结果；有些函数返回可选结果，可以赋值给标记或用在表达式中；有些函数返回的结果必须赋值给标记或用在表达式中。

要确定函数类型，可以看看函数描述。每个函数描述都有语法列表，显示该函数是否返回结果以及该结果是否为可选。

要调用不返回结果的函数

- ◆ 仅在语句中使用函数名（与参数，如果有）。例如：

```
FunctionName (Parameters);
```

要调用需要将结果赋值的函数

- ◆ 在脚本中任何可以使用值或相关数据类型的标记名的位置使用函数名（与参数，如果有）。例如，在赋值语句中：

```
ResultsTagname = FunctionName (Parameters);
```

或者在嵌套的函数调用中，将它用作其它函数的参数：

```
OtherFunction (FunctionName (Parameters));
```

要调用返回可选结果的函数

- ◆ 使用上述过程之一。

将参数传递给函数

标准预定义函数的参数通常通过 *值* 进行传递。这表示，只要表达式的求值结果属于参数所需的数据类型，便可以将任何有效的表达式作为参数进行传递。这类表达式可以包含值、标记引用及函数调用，所有这些都通过适当的运算符链接起来。如需有关表达式与运算符的详细信息，请参阅第 45 页的“赋值语句与运算符”。

脚本调用函数时，对表达式进行求值并将结果值传递给函数。

不过，有些函数需要将标记 *引用* 用作参数。例如：

```
RecipeSelectRecipe(Filename, RecipeName, Number);
```

在本例中，`RecipeName` 参数必须是标记引用（即，`RecipeName` 参数必须使用标记名）。即使该表达式通过求值可以得到有效的标记名，也不能使用传递字符串表达式的方式来代替。

备注 有些仅有一个参数的旧的预定义函数（例如，`Ack()` 函数）不遵循在括号中传递参数的标准语法。相反，参数使用空格同函数名进行分隔。如果对特定的函数有任何疑问，请查阅函数文档中的语法描述。

调用自定义函数 (QuickFunction)

调用自定义 QuickFunction 和调用预定义的标准函数略有不同：

- 关键字 CALL 必须加在 QuickFunction 名称之前。
- QuickFunction 返回的结果总是可选的；您可以使用它们，但并非必须如此。

要调用不返回结果的 QuickFunction

- ◆ 在语句中使用函数名（以及参数，如果有），前面加上关键字 CALL。例如：

```
CALL QuickFunctionName (Parameters);
```

要调用返回结果的 QuickFunction

- ◆ 执行以下操作之一：
 - 像不返回结果的那样调用 QuickFunction（请参阅上述过程）。
 - 在可以使用值或相关数据类型的标记名的脚本中的任何位置，使用函数名（以及参数，如果有），前面加上关键字 CALL。例如，在赋值语句中：

```
ResultsTagname = CALL  
QuickFunctionName (Parameters);
```

或者在嵌套的函数调用中，将它用作标准函数的参数：

```
OtherFunction (CALL FunctionName (Parameters));
```

备注 不能嵌套 QuickFunction 调用，让 QuickFunction 用作另一个 QuickFunction 的参数。例如，Call QF1(Call QF2()); 是无效的语句。

将参数传递给 QuickFunction

QuickFunction 的参数总是按值传递。不能通过引用将参数传递给 QuickFunction。

只要表达式进行求值可以得到参数所需的数据类型，就可以将任何有效的表达式作为参数进行传递。这类表达式可以包含值、标记引用及函数调用，所有这些都通过适当的运算符链接起来。如需有关表达式与运算符的详细信息，请参阅第 45 页的“赋值语句与运算符”。脚本调用函数时，对表达式进行求值并将结果值传递给函数。

备注 不能嵌套 QuickFunction 调用，让 QuickFunction 用作另一个 QuickFunction 的参数。例如，Call QF1(Call QF2()); 是无效的语句。

赋值语句与运算符

在脚本中，使用赋值语句将值写入标记。赋值语句的语法如下：

Tagname = ValueExpression;

执行此语句时，ValueExpression 写入由 Tagname 所引用的标记。ValueExpression 可以是数据类型与标记数据类型匹配的任何有效表达式。值表达式可以包含值、标记引用以及函数调用，所有这些都通过适当的运算符链接起来。

请参阅第 45 页的“支持的运算符”。

请参阅第 51 页的“设置运算符的求值顺序”。

请参阅第 53 页的“表达式的示例”。

支持的运算符

下表列出支持的所有运算符。如需有关使用特定运算符的详细信息，请参阅相关章节。

| 运算符 | 详细信息 |
|-----|-----------------------------|
| + | 第 46 页的“加法或串联：+” |
| - | 第 46 页的“减法或取反：-” |
| * | 第 46 页的“乘法：*” |
| / | 第 47 页的“除法：/” |
| ** | 第 47 页的“幂：**” |
| MOD | 第 47 页的“模除：MOD” |
| ~ | 第 47 页的“取补：~” |
| SHL | 第 47 页的“左移位：SHL 与右移位：SHR” |
| SHR | 第 47 页的“左移位：SHL 与右移位：SHR” |
| & | 第 48 页的“位与：&” |
| | 第 49 页的“位或： ” |
| ^ | 第 49 页的“位非：^” |
| AND | 第 49 页的“逻辑与：AND” |
| OR | 第 50 页的“逻辑或：OR” |
| NOT | 第 50 页的“逻辑非：NOT” |
| < | 第 51 页的“比较：<、>、<=、>=、==、<>” |
| > | 第 51 页的“比较：<、>、<=、>=、==、<>” |

| 运算符 | 详细信息 |
|-----|------------------------------|
| <= | 第 51 页的“比较: <、>、<=、>=、==、<>” |
| >= | 第 51 页的“比较: <、>、<=、>=、==、<>” |
| == | 第 51 页的“比较: <、>、<=、>=、==、<>” |
| <> | 第 51 页的“比较: <、>、<=、>=、==、<>” |

备注 对于数值计算，总是选择运算数，使得计算结果仍在“实数”的数值范围内。否则结果将不正确。

加法或串联: +

将两个数值运算数相加或串联两个字符串运算数。

有效运算数

加法: 任何“整型”或“实型”值

串联: 任何“消息”值

返回值的数据类型

加法: “整型”或“实型”

串联: “消息”

示例

```
MessageTag = "Setpoint value: " + Text(SetpointTag, "#.###");
```

减法或取反: -

使用两个运算数时，执行常规的数值减法。使用一个运算数时，切换运算数的符号。

有效运算数

任何“整型”或“实型”值

返回值的数据类型

“整型”或“实型”

示例

在本例中，如果 OriginalValue 是 70，InvertedValue 就变为 -70。如果 OriginalValue 是 -70，InvertedValue 就变为 70。

```
InvertedValue = -OriginalValue;
```

乘法: *

常规的数值乘法。

有效运算数

任何“整型”或“实型”值

返回值的数据类型

“整型”或“实型”

除法：/

常规的数值除法。如果试图在运行时除以 0，则 0 作为结果返回。

有效运算数

任何“整型”或“实型”值

返回值的数据类型

“整型”或“实型”

幂：**

将左侧运算数（基数）升为右侧运算数的幂（幂）。

有效运算数

“整型”或“实型”值。不能结合基数为 0、幂为负数，或基数为负、幂为小数的情况。在这些情况下，0 作为结果返回。

返回值的数据类型

“整型”或“实型”

示例

8 ** (1/3) 返回 2（8 的三次方根）

模除：MOD

返回两个整型值相除所得的余数。

有效运算数

任何“整型”值。

返回值的数据类型

整型

示例

37 MOD 4 返回 1

取补：~

返回整型值的补数。即，将每个 0 位转换为 1 位，反之亦然。

有效运算数

任何“整型”值。

返回值的数据类型

整型

左移位：SHL 与右移位：SHR

将整数值二进制表达式向左或向右移动指定的位数。左侧的运算数是要移动的值，右侧的运算数是位数。从字中移出的位会丢失。移走后腾空的位设为 0。

有效运算数

任何“整型”值。

返回值的数据类型

整型

示例

初始标记值 5 时，反复执行 `IntTag = IntTag SHL 1`；会得到以下结果：

| 迭代 | 二进制模式 | 标记值 |
|--------|----------------|-----|
| 初始值 | 0[...]00000101 | 5 |
| 执行 1 次 | 0[...]00001010 | 10 |
| 执行 2 次 | 0[...]00010100 | 20 |

位与：&

逐位比较两个整数的二进制表示法，然后按照下表返回结果：

| 第一个运算数中的位 | 第二个运算数中的位 | 结果中的位 |
|-----------|-----------|-------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

您可以使用这个运算符快速“屏蔽掉”（设为 0）位模式中特定的部分。例如，以下语句屏蔽掉 `IntTag` 标记的高 24 位：

`IntTag = IntTag & 255;`

如表中所示，如果运算数中的某个位是 0，则结果位总是 0。在 255 的二进制表示法中，只有低 8 位是 1，因此剩余的 24 个 0 位会使得结果中所有相应的位都设为 0。

有效运算数

任何“整型”值。

返回值的数据类型

整型

位或：|

逐位比较两个整数的二进制表示法，然后按照下表返回结果：

| 第一个运算数 中的位 | 第二个运算数 中的位 | 结果中的位 |
|---------------|---------------|-------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

此运算也称为“同或”。

有效运算数

任何“整型”值。

返回值的数据类型

整型

位非：^

逐位比较两个整数的二进制表示法，然后按照下表返回结果：

| 第一个运算数 中的位 | 第二个运算数 中的位 | 结果中的位 |
|---------------|---------------|-------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

此运算也称为“异或”。

有效运算数

任何“整型”值。

返回值的数据类型

整型

逻辑与：AND

如果两个离散运算数均为 TRUE（真），则返回 TRUE（真）；反之则返回 FALSE（假）。此运算符的真值表如下：

| p | q | p AND q |
|---|---|---------|
| F | F | F |
| F | T | F |
| T | F | F |

| p | q | p AND q |
|---|---|---------|
| T | T | T |

有效运算数

任何“离散”值。

返回值的数据类型

离散

逻辑或：OR

如果离散运算数中至少有一个为 TRUE（真），则返回 TRUE（真）；反之则返回 FALSE（假）。此运算符的真值表如下：

| p | q | p OR q |
|---|---|--------|
| F | F | F |
| F | T | T |
| T | F | T |
| T | T | T |

有效运算数

任何“离散”值。

返回值的数据类型

离散

逻辑非：NOT

如果离散运算数为 FALSE（假），则返回 TRUE（真），反之亦然。此运算符的真值表如下：

| p | NOT p |
|---|-------|
| F | T |
| T | F |

有效运算数

任何“离散”值。

返回值的数据类型

离散

比较：<、>、<=、>=、==、<>

这些运算符将两个值进行比较，如果符合运算符指定的条件，则返回 TRUE（真）。运算数可以是任何数据类型。对于字符串运算数，以不区分大小写基于字母表的顺序进行比较，b 大于 a，c 大于 b，等等。对于离散运算数，将 TRUE 视作大于 FALSE。下表列出所有比较运算符及其条件：

| 运算 | 示例 | 条件 |
|------|--------|-----------|
| 小于 | a < b | a 小于 b |
| 大于 | a > b | a 大于 b |
| 小于等于 | a <= b | a 小于或等于 b |
| 大于等于 | a >= b | a 大于或等于 b |
| 等于 | a == b | a 等于 b |
| 不等于 | a <> b | a 不等于 b |

有效运算数
任何数据类型值（两个值的数据类型必须相同）。
返回值的数据类型
离散

设置运算符的求值顺序

在任何表达式中，都可以使用括号来强制运算符以某种顺序进行求值。这同任何数学表达式中的工作方式相同。如果不使用括号，则表达式根据运算符的缺省优先规则进行求值。最高优先级的运算最先执行，第二高优先级的运算随后执行，依此类推。下表显示每个运算符的优先级。同一行上的运算符具有相同的优先级。

| | |
|--------------|-------|
| -、NOT、~ | 最高优先级 |
| ** | |
| *、/、MOD | |
| +, - | |
| SHL、SHR | |
| <, >, <=, >= | |
| ==, <> | |
| & | |
| ^ | |

| | |
|-----|-------|
| | |
| AND | |
| OR | |
| = | 最低优先级 |

隐式数据类型转换

InTouch HMI 脚本语言在赋值语句中某些特定的数据类型之间提供隐式数值转换。不过，这可能导致异常的结果，因此应小心使用此功能。

下表显示将某种类型的值指定给不同类型的标记时会发生什么。

| 期望的数据类型 | 使用的数据类型 | 附注 |
|---------|---------|------------------------------|
| 离散 | 整型 | 值 0 诠释为 FALSE。任何其它值诠释为 TRUE。 |
| 离散 | 实型 | 值 0 诠释为 FALSE。任何其它值诠释为 TRUE。 |
| 整型 | 离散 | 值 FALSE 转换为 0。值 TRUE 转换为 1。 |
| 整型 | 实型 | 只使用小数点之前的值。所有小数位均丢弃。 |
| 实型 | 离散 | 值 FALSE 转换为 0。值 TRUE 转换为 1。 |
| 实型 | 整型 | 值保留下来而不进行任何更改。 |

如需有关使用脚本函数在其它数据类型之间进行转换的详细信息，请参阅第 90 页的“转换数据类型”。

表达式的示例

下表显示一些有效的表达式，以及表达式的结果和结果的数据类型。

| 表达式 | 结果的数据类型 | 结果 |
|---|---------|---------------------------------|
| 37 MOD 4 | 整型 | 1 |
| 37 MOD 4 == 1 | 离散 | TRUE |
| NOT (37 MOD 4 == 1) | 离散 | FALSE |
| InfoAppActive(InfoAppTitle("xyz")) == 1 | 离散 | 如果 “xyz” 进程正在运行，则为 TRUE |
| "Batch " + Text(IntTag, "000") | 消息 | 如果 IntTag 的值为 10，则为 “Batch 010” |

下表显示一些无效的表达式及其无效的原因。

| 表达式 | 问题 |
|-------------------|--|
| NOT (37 MOD 4) | NOT 要求使用离散运算数。 |
| NOT 37 MOD 4 == 1 | NOT 的优先级比其它运算符的更高，因此 InTouch HMI 会试图给整数值 37 应用 NOT，而不是比较运算的离散结果。 |
| "Batch " + IntTag | 使用 + 运算符串联字符串时，两个运算数都必须是字符串。 |

使用条件程序分支结构

基于满足某些特定的条件，可以动态控制脚本的执行路径。为此，InTouch HMI 支持 IF-THEN-ELSE 控制结构。

IF-THEN-ELSE 控制结构的基本语法如下：

语法

```
IF Condition THEN
    ... 语句与 / 或另一个 IF-THEN-ELSE 结构
[ELSE
    ... 语句与 / 或另一个 IF-THEN-ELSE 结构 ]
ENDIF;
```

使用 IF-THEN-ELSE 结构时请记住以下规则：

- IF-THEN-ELSE 结构可以嵌套，都在 THEN 部分与 ELSE 部分进行。
- 对于每个 IF 语句，必有一个 ENDIF 结束语句。在同一嵌套级别中，ENDIF 语句总是应用于前面最接近的 IF 语句。
- Condition 必须是有效的离散表达式。如果 Condition 为 **TRUE**，则执行 THEN 部分。如果 Condition 为 **FALSE**，则执行 ELSE 部分。
- ELSE 部分是可选的。
- 其它一些编程语言可以在 IF-THEN-ELSE 结构的相同层级上检查多个条件，并有一个常规的 ELSE 部分，如果全部条件赋 **FALSE** 值，则执行这个部分。（Visual Basic 中的 If-ElseIf-Else 结构就是这样的例子）。在 InTouch HMI 中不能这样。对于要检查的每个条件，都必须开始一个新的 IF-THEN-ELSE 结构。因此，要让单个代码段用作所有条件的 ELSE 代码，必须将它放在 IF-THEN-ELSE 结构的最后一个嵌套级别的 ELSE 部分。

简单条件结构

以下脚本显示一个简单的条件结构。如果 SuccessTag 为 TRUE，则打开 "Success" 窗口，否则打开 "Failure" 窗口。

```
IF SuccessTag == 1 THEN
    Show "Success";
ELSE
    Show "Failure";
ENDIF;
```

嵌套条件结构

以下脚本显示如何检查多个条件并有一个常规 ELSE 部分，如果不满足任何条件则执行此部分代码。

```
IF ChoiceTag == 1 THEN
    Show "Procedure 1";
ELSE
    IF ChoiceTag == 2 THEN
        Show "Procedure 2";
    ELSE
        IF ChoiceTag == 3 THEN
            Show "Procedure 3";
        ELSE
            Show "Default Procedure";
        ENDIF;
    ENDIF;
ENDIF;
```

无效脚本示例（遗失 ENDIF）

如果熟悉 Visual Basic，可以尝试编写一个如下所示的简单 IF 语句：

```
IF OpenThisWindow == 1 THEN Show "This Window";
```

这在 InTouch HMI 中无效。对于每个 IF 语句，必有一个 ENDIF 结束语句。

无效脚本示例（嵌套不正确）

如果熟悉一种语言（如 Visual Basic），可能希望编写如下所示的带多个条件与一个缺省条件的条件结构：

```
IF ChoiceTag == 1 THEN
    Show "Procedure 1";
ELSE IF ChoiceTag == 2 THEN
    Show "Procedure 2";
ELSE IF ChoiceTag == 3 THEN
    Show "Procedure 3";
ELSE
    Show "Default Procedure";
ENDIF;
```

这在 InTouch HMI 中无效。每个 IF 都开始一个新的嵌套级别，且必须有相应的 ENDIF 语句。如需本示例的正确版本，请参阅第 55 页的“嵌套条件结构”。

使用程序循环

循环可以反复执行一段代码。InTouch HMI 仅支持 FOR 循环。FOR 循环按所监视的每次循环迭代所产生的递增或递减的数值循环变量值来进行。循环一直执行到循环变量值达到固定的极限。

语法

```
FOR LoopTag = StartExpression TO EndExpression [STEP  
    ChangeExpression]
```

... 语句或另一个 FOR 循环 ...

```
NEXT;
```

- StartExpression, EndExpression 与 ChangeExpression 共同定义迭代次数。
- StartExpression 设置循环范围的开始值。
EndExpression 设置循环范围的结束值。
- STEP ChangeExpression 可选择设置每次循环迭代过程中循环标记所递增或递减的值；如果不指定此值，则使用缺省值 1。

执行 FOR 循环时， InTouch HMI:

- 1 将 LoopTag 设置为 StartExpression 的值。
- 2 测试 LoopTag 是否大于 EndExpression。如果是， InTouch HMI 退出循环。（如果 If ChangeExpression 为负， InTouch HMI 测试 LoopTag 是否小于 EndExpression）。
- 3 执行循环内的语句。
- 4 按 ChangeExpression 的值（除非另外指定，否则设为 1）递增 LoopTag。
- 5 重复步骤 2 到 4。

使用 FOR 循环时请记住以下规则:

- FOR 循环可以嵌套。最大嵌套级数取决于可用的内存与系统资源。
- 对于每个 FOR 语句，必有一个 ENDIF 结束语句。在同一嵌套级别中， NEXT 语句总是应用于前面最接近的 FOR 语句。
- LoopTag 必须是数值标记（或局部变量）。
- StartExpression、 EndExpression 以及 ChangeExpression 必须是赋值为数值结果的有效表达式。
- 如果 ChangeExpression 为正， EndExpression 必须大于 StartExpression；如果 ChangeExpression 为负， StartExpression 必须大于 EndExpression。否则循环不会开始。
- 要退出循环，请使用 EXIT FOR 语句。如需有关详细信息，请参阅第 58 页的“强制结束循环”。
- 循环有时间限制。请参阅第 59 页的“循环执行的时间限制”。

注意 循环的执行会影响其它运行时进程。如需有关详细信息，请参阅第 58 页的“循环对其它运行时进程的影响”。

强制结束循环

您可以通过调用以下语句在任何时间退出循环：

```
EXIT FOR;
```

此语句使脚本继续执行紧接着循环 NEXT 语句之后的语句。

示例

下面的代码段使用循环将大量的虚拟记录插入数据库表。如果插入记录时发生错误，则放弃循环以防止产生更多错误。

```
FOR Counter = 1 TO 1000
    ResultCode = SQLInsert(ConnectionID, "BatchDetails",
        "BindList1");
    IF ResultCode <> 0 THEN
        LogMessage("Error creating records!
            Aborting...");
        EXIT FOR;
    ENDIF;
NEXT;
```

循环对其它运行时进程的影响

执行 FOR 循环时，WindowViewer 中的所有其它运行时进程都暂停。这包括以下范围：

- 屏幕更新（动画链接、值显示、趋势等）。这表示，到循环完成之前不会发生任何移动，因此不能给动画对象使用 FOR 循环。
- I/O 通讯。例如，如果修改 FOR 循环中 I/O 标记的值，则只有最终的迭代后面的值才会写入 I/O 设备。
- 其它脚本，包括异步 QuickFunction。

您可以通过将 FOR 循环放入异步 QuickFunction 来避免暂停其它运行时进程。

循环执行的时间限制

为避免无限循环，这里设置了一个时间限制，FOR 循环必须在这个时间限制内完成执行。如果循环未在这个时间跨度之后完成，WindowViewer 会自动终止循环它，并将一条关于终止的消息写入 Log Viewer。

缺省的时间限制为 5 秒。通过向应用程序目录中的 intouch.ini 文件添加下面这行，可以对它进行自定义：

```
LoopTimeout=x
```

将 x 替换成以秒为单位的时间限制。

备注 时间限制仅在循环的 NEXT 语句中检查。因此，循环的第一次迭代总是会执行，即便它花费的时间比时间限制更长。

循环的示例

以下脚本通过一个简单的循环与一个间接标记，使用 0 值来重新初始化 100 个标记（Tag001 到 Tag100）。

```
DIM Counter AS INTEGER;
FOR Counter = 1 TO 100
    IndirectInteger.Name = "Tag" + Text(Counter, "000");
    IndirectInteger.Value = 0;
NEXT;
```

以下脚本通过两个嵌套的循环与一个间接标记，使用 0 来重新初始化 1000 个标记（Line01_Tag001 到 Line10_Tag100）。

```
DIM LineCounter AS INTEGER;
DIM TagCounter AS INTEGER;
FOR LineCounter = 1 TO 10
    FOR TagCounter = 1 TO 100
        IndirectInteger.Name = "Line" +
            Text(LineCounter, "00") + "_Tag" +
            Text(TagCounter, "000");
        IndirectInteger.Value = 0;
    NEXT;
NEXT;
```

使用局部变量

您可以在脚本中声明多个局部变量，以存储临时或中间结果。这可以提高性能并帮助维持低标记计数。您可以在脚本中像使用标记名那样使用局部变量。不过有些不同之处：

- 局部变量仅存在于声明它们的脚本的范围内。在脚本执行完毕时，它们的值会丢失。它们不能由应用程序中的任何其它脚本引用。
- 局部变量没有点域。
- 局部变量不计入标记计数。

可以在脚本中使用局部变量之前，必须先声明它；否则会将引用视作标记名。请参阅第 60 页的“声明局部变量”。

您可以声明与标记使用相同名称的局部变量。请参阅第 61 页的“局部变量与标记之间的命名冲突”。

声明局部变量

您可以在脚本中的任何位置声明局部变量，只要在第一次使用它们之前进行声明即可。要声明局部变量，请使用以下语句：

```
DIM LocVarName [AS DataType];
```

LocVarName 为局部变量的名称。名称必须符合标记名的命名惯例。如需有关详细信息，请参阅 *InTouch® HMI 数据管理指南* 中的第 2 章“使用标记名字典管理标记”中的“标记名惯例”。

DataType 是局部变量的数据类型。有效值是离散、整型、实型以及消息。如果不指定此选项，则缺省使用整型。

对于要声明的每个局部变量，必须使用一个单独的 DIM 语句。

您可以声明任何数量的局部变量。数量仅受可用内存的限制。

示例

要声明整型变量：

```
DIM MyLocalIntVar AS Integer;
```

要声明多个实型变量：

```
DIM MyLocalRealVar1 AS Real;  
DIM MyLocalRealVar2 AS Real;
```

以下语句无效：

```
DIM MyLocalRealVar1, MyLocalRealVar2 AS Real;
```

局部变量与标记之间的命名冲突

您可以使用与现有标记相同的名称来声明局部变量。不过，在脚本中引用该名称时，局部变量总是比标记优先。例如，假设有一个现有的“整型”标记“iTag”，并运行以下脚本：

```
DIM iTag as Integer;
```

```
iTag = 20;
```

在这种情形中，赋值语句仅将一个值写入局部变量。同名标记的值保持不变。

第 5 章

自定义脚本函数

InTouch HMI QuickFunction 是其它环境中可能被称为宏、子程序或过程的脚本。

关于 QuickFunction

QuickFunction 是可以从其它脚本与动画链接中调用的脚本。QuickFunction 的主要优点是减少重复代码。

您可以将值传递给可以使用这些值并返回结果的 QuickFunction。

QuickFunction 可以异步运行。与其它脚本不同，它们可以在后台运行，而不会干扰主程序流程。异步运行的 QuickFunction 可用于耗时的操作，如 SQL 数据库调用。

备注 仔细设计 QuickFunction 及其参数，因为如果希望修改 QuickFunction 中的参数，必须先从使用 QuickFunction 的每个脚本中删除对该 QuickFunction 的所有调用。进行更改之后，必须再将 QuickFunction 调用添加回这些脚本中。请参阅第 64 页的“配置 QuickFunction”中的备注。

QuickFunction 有三个基本部分：

- 名称
- 参数（可选）
- 带可选返回值的脚本主体

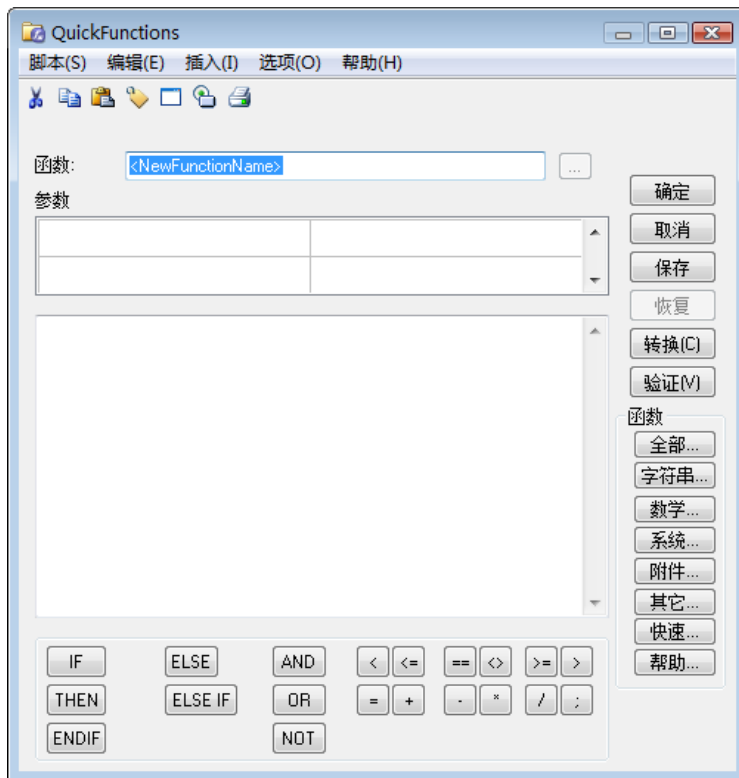
QuickFunction 通过在动画链接或其它脚本中使用 CALL 函数来执行。请参阅第 66 页的“调用 QuickFunction”。

配置 QuickFunction

您可以创建、修改或删除 QuickFunction。

要创建 QuickFunction

- 1 在脚本窗格中，使用鼠标右键单击 **QuickFunctions**，然后单击**新建**。此时出现 **QuickFunctions** 对话框。



- 2 在函数框中，输入 QuickFunction 的名称。

- 3 在**参数**区域中，为每个参数在左侧输入名称、在右侧输入数据类型。

这些参数是局部变量，它们仅存在于定义它们的 QuickFunction 中。每个 QuickFunction 最多可包含 16 个参数。参数名可以包含 31 个字符，但不能是空格。参数名必须以英文字母开头。参数名必须是唯一的。

- 4 在窗口中输入脚本。
- 5 要使 QuickFunction 返回结果，请将以下内容添加到脚本中：RETURN *值*
*值*可以是值、局部变量、全局标记名或计算表达式。脚本在 RETURN 命令处终止，在调用函数处继续。
- 6 单击**确定**。

要修改 QuickFunction

- 1 在**脚本**窗格中，展开 **QuickFunctions**，使用鼠标右键单击要修改的 QuickFunction，然后单击**编辑**。此时出现 **QuickFunctions** 对话框。
- 2 对脚本主体进行修改，然后单击**确定**。

备注 如果在 InTouch 应用程序中存在对 QuickFunction 的调用，则不能修改参数列表。您必须先删除这些调用，关闭所有 InTouch 窗口，并更新使用计数。

要删除 QuickFunction

- 1 删除对 QuickFunction 的所有调用，关闭所有 InTouch 窗口，并更新使用计数。
- 2 在**脚本**窗格中，展开 **QuickFunctions**，使用鼠标右键单击要删除的 QuickFunction，然后单击**删除**。出现消息时，单击**是**。

调用 QuickFunction

您可以配置脚本与动画链接，以调用 **QuickFunction** 并处理或显示可能的返回值。

如果参数值没有更改，则不会调用 **QuickFunction**。您可以使用 **\$second** 作为参数，以确保至少每秒执行一次 **QuickFunction**。

如需有关详细信息，请参阅第 44 页的“调用自定义函数 (**QuickFunction**)”。

创建异步 QuickFunction

您可以定义 **QuickFunction** 同主程序流程异步（即，并行）运行。

要创建异步 QuickFunction

- 1 在“脚本编辑器”中，创建 **QuickFunction**。
- 2 在选项菜单上，单击**异步的**。

异步 QuickFunction 的限制

您无法：

- 从异步 **QuickFunction** 中返回某个值。
- 同时运行相同 **QuickFunction** 的多个实例。
- 在 **QuickFunction** 开始执行异步之后将它们停止。

您不应：

- 同时运行三个以上不同的异步 **QuickFunction**。同时运行三个以上 **QuickFunction** 会大幅降低系统性能。
- 将异步函数用作动画链接表达式的一部分，如工具提示。

检查是否有任何异步 QuickFunction 正在运行

您可以使用 `IsAnyAsyncFunctionBusy()` 函数检查是否有任何异步 QuickFunction 正在运行。您可以使用此函数使调用异步 QuickFunction 的 QuickScript 等待所有其它异步 QuickFunction 完成处理。

`IsAnyAsyncFunctionBusy()` 函数

返回一个离散值，指出是否有任何异步 QuickFunction 正在运行。

语法

`result = IsAnyAsyncFunctionBusy (timeout)`

参数

result

离散值，指出是否有任何异步 QuickFunction 正在运行，含义如下：

- 0 = 没有异步 QuickFunction 正在运行。
- 1 = 有异步 QuickFunction 正在运行。

timeout

检查是否有任何异步 QuickFunction 正在运行之前要等待的秒数。整数值、整型标记名或整型表达式。

示例

假设要使用异步 QuickFunction 连接到多个 SQL 数据库，并且知道要 2 分钟才能建立这些连接。

首先，执行异步 QuickFunction 连接到 SQL 数据库。

接着，使用 QuickScript 中的 `IsAnyAsyncFunctionBusy(120)` 函数，让 SQL 有足够的时间在该 QuickFunction 完成之前建立这些连接。

如果在 2 分钟之后这些连接仍未建立，且异步 QuickFunction 仍忙于建立这些连接，则函数 `IsAnyAsyncFunctionBusy()` 返回值 1 (true)。

现在可以显示一条错误消息，告诉操作员 SQL 连接不成功。

以下脚本实现了这种情形的处理：

```
IF IsAnyAsyncFunctionBusy(120) == 1 THEN
    SHOW "SQL Connection Error Dialog";
ENDIF;
```

停止运行异步 QuickFunction

在异步 QuickFunction 开始之后无法将其停止，但可以通过停止脚本逻辑来停止启动其它的异步 QuickFunction。这会影响 InTouch 应用程序中的所有 QuickScript。

如需有关停止脚本执行的详细信息，请参阅第 37 页的“在运行时暂停脚本执行”。

第 6 章

内置函数

InTouch QuickScript 函数可以在满足指定条件的情况下执行命令与逻辑运算。您可以单独使用 QuickScript 函数并在满足特定条件时执行它们，也可以在动画显示链接中使用它们。

在动画显示链接中强制更新

如果在动画链接中使用 QuickScript，则仅当标记与这些动画链接关联时，它们才会更新。只要标记的值发生改变，此标记便充当触发器。使用 \$Second 或 \$Minute 系统标记来更新动画链接是个较好的选择。

要在动画显示链接中进行强制更新

- 1 在对象属性窗口中打开动画链接。
- 2 将触发器标记（例如 \$Second）添加到计算公式中。例如：
 - 如果动画链接是实型或整型，则可以使用 \$Second/\$Second 乘以表达式。
 - 如果动画链接是字符串，则可以将 StringMid(\$TimeString, 0, 0) 添加到表达式。
 - 如果动画链接是离散型，则可以将 (\$second.00 - \$second.00) 添加到表达式。

数学计算

InTouch HMI 支持基本数学函数，这些函数可以用在脚本与动画链接中，例如具有以下用途的函数：

- 舍入与截断数字。
- 计算正弦和余弦。
- 计算对数与指数。
- 计算平方根。

舍入、截断及确定符号

在脚本中，可以使用以下函数来舍入数字、截断数字以及确定数字的符号：

| 函数 | 作用 |
|---------|----------------------|
| Abs() | 计算某个值或表达式的绝对值。 |
| Int() | 计算某个值或表达式的整数部分。 |
| Round() | 舍入某个值或表达式。 |
| Sgn() | 确定某个值或表达式的符号（负、正、零）。 |
| Trunc() | 返回某个值或表达式小数点前面的部分。 |

Abs() 函数

返回指定的数字的绝对值。您可以使用此函数将负数转换为正数。

语法

```
result = Abs (number)
```

参数

number

数字、模拟标记名或数值表达式。

示例

Abs (14) 返回 14。

Abs (-7.5) 返回 7.5。

Int() 函数

返回小于（或等于）指定数字的整数。

语法

result = Int (*number*)

参数

number

数字、模拟标记名或数值表达式。

示例

Int (4.7) 返回 4。

Int (-4.7) 返回 -5。

备注 对于负实数，此函数将返回小于指定数字的整数。例如，Int(-4.7) 不是 -4，而是 -5。要返回整数部分，请使用 Trunc() 函数。请参阅第 72 页的“Trunc() 函数”。

Round() 函数

将数字舍入到指定的精度。结果是实数。

语法

result = Round (*number*, *precision*)

参数

number

数字、模拟标记名或数值表达式。

precision

数字所要舍入的精度。它可以是数字、模拟标记名或数值表达式。

示例

Round (4.3, 1) 返回 4。

Round (4.3, 0.01) 返回 4.30。

Round (4.5, 1) 返回 5。

Round (-4.5, 1) 返回 -4。

Round (106, 5) 返回 105。

Round (43.7, 0.5) 返回 43.5。

Sgn() 函数

返回某个数字的符号。使用它来确定数字、标记名或表达式是负、正还是零。

语法

```
result = Sgn (number)
```

参数

number

数字、模拟标记名或数值表达式。

示例

Sgn(425) 返回 1。

Sgn(0) 返回 0。

Sgn(-37.3) 返回 -1。

Trunc() 函数

返回数字被截断后的值。截断后的值是小数点前面的部分。使用它得到实数的整数部分。

语法

```
result = Trunc (number)
```

参数

number

数字、模拟标记名或数值表达式。

示例

Trunc(4.3) 返回 4。

Trunc(-4.3) 返回 -4。

备注 您也可以使用此函数来处理数字的小数部分。要返回指定数字的小数部分，请按以下方式使用 Trunc() 函数：

```
result = number - trunc(number);
```

使用三角函数

在脚本中，可以使用以下函数来进行三角计算。

| 函数 | 作用 |
|----------|----------------|
| Sin() | 计算某个角度的正弦。 |
| ArcSin() | 计算某个值或表达式的反正弦。 |
| Cos() | 计算某个角度的余弦。 |
| ArcCos() | 计算某个值或表达式的反余弦。 |
| Tan() | 计算某个角度的正切。 |
| ArcTan() | 计算某个值或表达式的反正切。 |

备注 InTouch HMI 中的 QuickScript 三角函数使用以度 (0 - 360) 表示的角度。要使用弧度取代之，必须在将参数传递给函数之前，或从函数检索结果之后，执行相应的计算。

Sin() 函数

返回某个数字的正弦。对于三角函数而言，该数字是以度表示的角度。

语法

result = Sin (*number*)

参数

number

数字、模拟标记名或数值表达式。

示例

Sin(90) 返回 1。

Sin(0) 返回 0。

Sin(30) 返回 0.5。

100 * Sin (6 * \$second) 返回振幅为 100、周期为 1 分钟的正弦波。

ArcSin() 函数

返回某个数字的反正弦。它是 Sin() 函数的反函数。使用 ArcSin() 函数计算介于 -90 和 90 度之间的角度，其正弦等于该数字。

语法

```
result = ArcSin (number)
```

参数

number

-1 到 1 范围内的一个数字、模拟标记名或数值表达式。

示例

ArcSin(1) 返回 90。

ArcSin(0) 返回 0。

ArcSin(0.5) 返回 30。

Cos() 函数

返回某个数字的余弦。对于三角函数而言，该数字是以度表示的角度。

语法

```
result = Cos (number)
```

参数

number

数字、模拟标记名或数值表达式。

示例

Cos(90) 返回 0。

Cos(0) 返回 1。

Cos(60) 返回 0.5。

20 + 50 * Cos(6 * %second) 产生振幅为 50、周期为一分钟围绕 20 振荡的正弦波。

ArcCos() 函数

返回数字的反余弦。它是 Cos() 函数的反函数。使用 ArcCos() 函数计算介于 0 到 180 度之间的角度，其余弦等于该数字。

语法

```
result = ArcCos (number)
```

参数

number

-1 到 1 范围内的一个数字、模拟标记名或数值表达式。

示例

ArcCos(1) 返回 0。

`ArcCos(-0.5)` 返回 120。

Tan() 函数

返回指定数字的正切。对于三角函数而言，该数字是以度表示的角度。

语法

`result = Tan (number)`

参数

number

数字、模拟标记名或数值表达式。

示例

`Tan(45)` 返回 1。

`Tan(0)` 返回 0。

ArcTan() 函数

返回某个数字的反正切。它是 `Tan()` 函数的反函数。使用 `ArcTan()` 函数计算其正切等于该数字的角度。

语法

`result = ArcTan (number)`

参数

number

数字、模拟标记名或数值表达式。

示例

`ArcTan(1)` 返回 45。

`ArcTan(0)` 返回 0。

返回 Pi 的值

在脚本中，可以使用 `Pi()` 函数在数学计算中使用常量 *Pi*。 `Pi()` 函数精确到小数点后 7 位。

语法

`result = Pi ()`

示例

`Pi()` 返回 3.1415927。

计算对数

在脚本中，可以使用以下函数来运行使用对数和指数函数的计算。

| 函数 | 作用 |
|--------|-------------------------|
| Log() | 计算某个值或表达式的自然对数。 |
| Exp() | 计算某个值或表达式的指数。 |
| LogN() | 计算以另一个值或表达式为底的值或表达式的对数。 |

Log() 函数

返回指定正数的自然对数。这是 Exp() 函数的反函数。

备注 0 与负数的自然对数未定义。如果将 0 或负数传递给 Log() 函数，则返回 -99.0000000。

语法

```
result = Log (number)
```

参数

number

正数、模拟标记名或数值表达式。

示例

Log(100) 返回 4.6051702。

Log(1) 返回 0。

Exp() 函数

返回指定数字的指数。这是 Log() 函数的反函数，它等于 e 的幂。

备注 如果将 -88.72 与 88.72 范围之外的值传递给 Exp() 函数，则返回 -99.0000000。

语法

```
result = Exp (number)
```

参数

number

-88.72 到 88.72 范围内的一个数字、模拟标记名或数值表达式。

示例

Exp(1) 返回 2.7182818。

Exp(0) 返回 1。

LogN() 函数

返回指定底的正数的对数。这是底的对数次幂的反函数。

示例

语法

result = LogN (number, base)

参数

number

正数、模拟标记名或数值表达式。

base

不等于 1 的正数、模拟标记名或表达式。

示例

LogN(8,2) 返回 3。

LogN(num,btag) 返回以 btag 为底的 num 的对数。

备注 如果将无效的参数传递给 LogN() 函数，则返回 -99.0000000。

计算平方根

在脚本中，可以使用 Sqrt() 函数计算指定的非负数的平方根。

备注 如果将负值传递给 Sqrt() 函数，则返回 -99.0000000。

语法

result = Sqrt (number)

参数

number

非负的数字、模拟标记名或数值表达式。

示例

Sqrt(36) 返回 6。

Sqrt(perftag) 返回标记名 perftag 的值的平方根。

字符串运算

您可以在脚本与动画链接中使用许多基本字符串函数。您可以使用这些函数：

- 返回字符串的某些部分。
- 更改字符串大小写。
- 给字符串删除与添加空格。
- 处理字符串中的 ASCII 值。
- 在字符串中搜索并替换。
- 相互比较字符串。
- 返回有关字符串的其它信息，如长度。

返回字符串的某些部分

在脚本中，可以使用 `StringLeft()`、`StringMid()` 以及 `StringRight()` 函数返回字符串的某些部分。

StringLeft() 函数

从字符串的开头起返回指定数量的字符。

语法

```
result = StringLeft (string, length)
```

参数

string

文本、消息标记名或字符串表达式。

length

返回的字符数量。数字、模拟标记名或数值表达式。

示例

```
StringLeft("Hello World",5) 返回 “Hello”。
```

```
StringLeft("Hello World",20) 返回 “Hello World”。
```

```
StringLeft("Hello World".0) 返回 “Hello World”。
```

备注 如果将 0 作为 `length` 传递给 `StringLeft()` 函数，则返回整个字符串。

StringRight() 函数

从字符串的末尾起返回指定数量的字符。

语法

```
result = StringRight (string, length)
```

参数

string

文本、消息标记名或字符串表达式。

length

要返回的字符数。数字、模拟标记名或数值表达式。

示例

StringRight("Hello World",5) 返回 “World”。

StringRight("Hello World",20) 返回 “Hello World”。

StringRight("Hello World".0) 返回 “Hello World”。

备注 如果将 0 作为 length 传递给 StringRight() 函数，则返回整个字符串。

StringMid() 函数

返回字符串的一部分。您可以指定开始点与要返回的字符数。

语法

```
result = StringMid (string, startpos, length)
```

参数

string

文本、消息标记名或字符串表达式。

startpos

字符串中的开始位置。数字、模拟标记名或数值表达式。

length

要返回的字符数。数字、模拟标记名或数值表达式。

示例

StringMid("Hello World",5,4) 返回 "o Wo”。

StringMid("Hello World",7,50) 返回 “World”。

StringMid("Hello World",4,0) 返回 “lo World”。

备注 如果将 0 作为 length 传递给 StringMid() 函数，则返回开始位置之后的整个字符串。

更改字符串大小写

在脚本中，可以使用 `StringLower()` 与 `StringUpper()` 函数以小写或大写形式返回指定的字符串。通过将结果指定给指定的字符串，可以将大写转换为小写，反之亦然。

StringLower() 函数

返回字符串相应的小写形式。

语法

```
result = StringLower (string)
```

参数

string

文本、消息标记名或字符串表达式。

示例

`StringLower("TURBINE")` 返回 “turbine”。

`StringLower("The Value Is 22.2")` 返回 “the value is 22.2”。

`mtag = StringLower(mtag)` 将 `mtag` 的消息值转换为小写形式。

StringUpper() 函数

返回字符串相应的大写形式。

语法

```
result = StringUpper (string)
```

参数

string

文本、消息标记名或字符串表达式。

示例

`StringUpper("abcd")` 返回 “ABCD”。

`StringUpper("The Value Is 22.2")` 返回 “THE VALUE IS 22.2”。

`mtag = StringUpper(mtag)` 将 `mtag` 的消息值转换为大写形式。

从字符串中删除空格

在脚本中，可以使用 **StringTrim()** 函数从字符串中修剪掉前导与尾部空格（空白符）。您可以使用此函数在用户输入之后（举例而言）从字符串中删除多余的空格。

StringTrim() 函数

从字符串中修剪掉前导与尾部空格（空白）。您可以使用此函数在用户输入之后（举例而言）从字符串中删除多余的空格。

语法

```
result = StringTrim (string, trimtype)
```

参数

string

文本、消息标记名或字符串表达式。

trimtype

确定要删除的空格的值、模拟标记名或数值表达式：

- 1 = 前导空格。
- 2 = 尾部空格。
- 3 = 前导与尾部空格。

附注

此函数从字符串中删除所有的前导与尾部空白。空白是空格 (ASCII 0x20) 与 ASCII 0x09 到 0x0D 之间的控制字符。

示例

要删除包含动作脚本的消息标记 **mtag** 中的所有空格，使用以下脚本：

```
DIM i AS INTEGER;
DIM tmp AS MESSAGE;
mtag = StringTrim(mtag,3);  {mtag is trimmed}
FOR i = 1 TO StringLen(mtag)  {run variable i over the characters of
    mtag}
    IF StringMid(mtag, i, 1)<>" " THEN  {i-th character is not space}
        tmp = tmp + StringMid(mtag, i, 1);  {
add that character to tmp}
    ENDIF;
NEXT;
mtag = tmp;  {pass tmp back to mtag}.
```

其它示例：

`StringTrim(" Joe ",1)` 返回 “Joe”。

`StringTrim(" Joe ".2)` 返回 “Joe”。

此脚本删除 **mtag** 值左侧与右侧的所有空格：

```
mtag = StringTrim(mtag, 3)
```

使用空格设置字符串格式

在脚本中，可以使用 `StringSpace()` 函数将空格（空白）添加到字符串。

语法

```
result = StringSpace(number)
```

参数

number

数字、数值标记名或数值表达式。

示例

`StringSpace(4)` 返回包含 4 个空格的字符串。

`"Pump"+StringSpace(1)+"Station"` 返回 “Pump Station”。

在字符与 ASCII 码之间转换

在脚本中，通过使用 `StringChar()` 与 `StringASCII()` 函数，可以将字符串中的字符转换成 ASCII 码，也可以将 ASCII 码转换成字符。

这些函数不支持多字节字符集。仅支持范围在 0-255 的字符。

如果希望在字符串上执行某些数值计算（例如，对字符串进行编码），则使用 ASCII 码非常有用。

StringChar() 函数

返回指定的 ASCII 码对应的单个字符。

语法

```
result = StringChar(ASCIICode)
```

参数

ASCIICode

介于 0 到 255 之间的一个数字、数值标记名或数值表达式。

附注

对于将控制字符传递给外设（如打印机或调制解调器）或将英文双引号传递给 SQL 查询，此函数非常有用。

示例

`StringChar(65)` 返回 "A"。

此脚本返回用英文双引号括起的 “Hello World”：

```
StringChar(34)+"Hello World"+StringChar(34)
```

此脚本返回用回车换行符将两个词分开来的 “Hello World”：

```
"Hello"+StringChar(13)+StringChar(10)+"World"
```

StringASCII() 函数

返回字符串第一个字符的 ASCII 码。

语法

```
result = StringASCII (string)
```

参数

string

字符串、消息标记名或字符串表达式。

示例

StringASCII("A") 返回 65。

StringASCII("hello world") 返回 104。

搜索与替换字符串中的文本

对于使用单字节字符集的语言（如英语），可以使用脚本中的 StringInString() 与 StringReplace() 函数在消息标记上执行有限的搜索与替换功能。

| 函数 | 作用 |
|------------------|--|
| StringInString() | 在另一个字符串中搜索特定的字符串，将位置作为结果返回。 |
| StringReplace() | 在指定的字符串中使用其它字符或字替换特定的字符或字，将新字符串作为结果返回。 |

StringInString() 函数

返回指定的字符串在另一个字符串中出现的第一个位置。

语法

```
result = StringInString (string, searchfor, startpos, casesens)
```

参数

string

这是要被搜索的字符串。字符串、消息标记名或字符串表达式。

searchfor

这是要搜索的字符串。字符串、消息标记名或字符串表达式。

startpos

这是字符串中搜索的起始位置。值、数值标记名或数值表达式。

casesens

确定搜索是否区分大小写。可以是 0 或 1、离散标记名或布尔表达式。

0 - 搜索不区分大小写（大写和小写形式视为相同）。

1 - 搜索区分大小写（大写和小写形式视为不同）。

附注

使用此函数来确定是否有特定的字符串包含在消息标记中。您可以指定搜索的起始位置以及是否考虑字母大小写。

示例

此脚本返回 5 — 由于 “MTX” 中的首字母 “M” 位于字符串中的第 5 个位置：

```
StringInString("DBO MTX-010","MTX",1,0)
```

此脚本返回 3 — 由于 “MTX” 中的首字母 “M” 位于字符串中的第 3 个位置：

```
StringInString("T-MTX 010 MTX","MTX",1,0)
```

此脚本返回 11 — 由于第 8 个位置之后的 “MTX” 中的首字母 “M” 位于字符串中的第 11 个位置：

```
StringInString("T-MTX 010 MTX","MTX",8,0)
```

此脚本返回 11 — 由于同 MTX 匹配且大小写一致的第一个字符串在第 11 个位置：

```
StringInString("t-mtx 030 MTX", "MTX",1,1)
```

此脚本返回 0 — 由于字符串中没有 “Mty”：

```
StringInString("t-mtx 030 MTY-Mtx", "Mty",1,1)
```

StringReplace() 函数

在一个字符串中搜索另一个字符串，如果找到，则使用第三个字符串来替换它。您可以指定：

- 区分大小写 - 这确定是否将大写字母与小写字母视为完全相同的字母。
- 要替换的次数 - 如果搜索字符串出现多次，这会非常有用。
- 全字匹配 - 如果搜索字符串是整个单词，则使用这项。

备注 此函数不支持双字节字符集。

语法

```
result = StringReplace (string, searchfor, replacewith, casesens,
                        numtoreplace, matchwholewords)
```

参数

string

要被搜索的字符串。字符串、消息标记名或字符串表达式。

searchfor

要搜索的字符串。字符串、消息标记名或字符串表达式。

replacewith

要用作替换的字符串。字符串、消息标记名或字符串表达式。

casesens

确定搜索是否区分大小写。可以是 0 或 1、离散标记名或布尔表达式。

0 - 搜索不区分大小写（大写和小写形式视为相同）

1 - 搜索区分大小写（大写和小写形式视为不同）

numtoreplace

要替换的量。将其设置为 -1 以替换找到的所有搜索字符串。整数值、整型标记名或整型表达式。

matchwholewords

确定是否仅限全字匹配。可以是 0 或 1、离散标记名或布尔表达式。

0 - 此函数查找字符串中任何位置的搜索字符串字符

1 - 仅先全字匹配

示例

此语句仅替换第一项，并返回 "MTY 030 MTX"。

```
StringReplace("MTX 030 MTX", "MTX", "MTY", 0, 1, 0)
```

此语句替换所有项，并返回 "MTY 030 MTY"。

```
StringReplace("MTX 030 MTX", "MTX", "MTY", 0, -1, 0)
```

此语句替换大小写匹配的所有项，并返回 "MTY 030 mtx"。

```
StringReplace("MTX 030 mtx", "MTX", "MTY", 1, -1, 0)
```

此语句替换全字匹配的所有项，并返回 "MTY 030 QMTX"。

```
StringReplace("MTX 030 QMTX", "MTX", "MTY", 0, -1, 1)
```

返回关于字符串的信息

在脚本中，通过使用 `StringLen()` 与 `StringTest()` 函数可以返回指定字符串的长度，以及测试某个字符是否在特定的字符组中。

StringLen() 函数

返回指定字符串（包括不可见字符）的长度。

语法

```
result = StringLen (string)
```

参数

string

字符串、消息标记名或字符串表达式。

示例

`StringLen("Twelve percent")` 返回 14。

`StringLen("12%")` 返回 3。

`StringLen("The end." + StringChar(13))` 返回 9。

StringTest() 函数

测试字符串的第一个字符是否在特定的字符组中。

语法

```
result = StringTest (string, group)
```

参数

string

字符串、消息标记名或字符串表达式。

group

要测试字符的组数。1 到 11 范围内的值、整型标记名，或整型表达式。

- 1 - 字母数字字符 (A-Z、a-z、0-9)
- 2 - 数值字符 (0-9)
- 3 - 字母字符 (A-Z、a-z)
- 4 - 大写字符 (A-Z)
- 5 - 小写字符 (a-z)
- 6 - 标点字符 (ASCII 0x21 - 0x2F)，例如 !、@、#、\$、%、^、&、* 等
- 7 - ASCII 字符 (ASCII 0x00 - 0x7F)
- 8 - 十六进制字符 (0-9、A-F、a-f)
- 9 - 可打印字符 (ASCII 0x20 - 0x7E)
- 10 - 控制字符 (ASCII 0x00 - 0x1F 与 0x7F)
- 11 - 空格字符 (ASCII 0x09 - 0x0D 与 0x20)

示例

此字符串返回 1— 由于 “A” 是字母数字字符：

```
StringTest("ACB123",1)
```

此字符串返回 0— 由于 “A” 不是小写字符：

```
StringTest("ABC123",5)
```

比较字符串

在脚本中，通过使用 `StringCompare()`、`StringCompareNoCase()` 以及 `StringCompareEncrypted()` 函数，可以比较两个字符串。

| 函数 | 作用 |
|---------------------------------------|-------------------|
| <code>StringCompare()</code> | 比较时区分大小写。 |
| <code>StringCompareNoCase()</code> | 比较时不区分大小写。 |
| <code>StringCompareEncrypted()</code> | 比较加密的字符串和未加密的字符串。 |

`StringCompare()` 函数

相互比较两个字符串，并返回布尔结果（0 = 字符串相等）。每个字母的大小写也在考虑范围，因此举例而言，‘A’ 会视为不等于 ‘a’。

语法

```
result = StringCompare (string1, string2)
```

参数

string1

字符串、消息标记名或字符串表达式。

string2

字符串、消息标记名或字符串表达式。

示例

```
StringCompare ("Apple", "Apple") 返回 0。
```

```
StringCompare ("Apple", "apple") 返回 1。
```

此字符串比较两个消息标记，并返回离散结果（0 或 1）：

```
StringCompare (mtag1, mtag2)
```


StringCompareNoCase() 函数

相互比较两个字符串，并返回整型结果。每个字母的大小写不在考虑范围，因此举例而言，‘A’会视作等于‘a’。

整型结果返回：

- 0，如果两个字符串完全相同（忽略大小写）。
- 非零值，其它情况。结果是不同字符的 ASCII 值之差（忽略大小写）。

备注 由于在 InTouch 脚本中，所有非零值都视作等于 TRUE，StringCompareNoCase() 函数的结果可以作为离散结果来使用。

语法

result = StringCompareNoCase (*string1*, *string2*)

参数

string1

字符串、消息标记名或字符串表达式。

string2

字符串、消息标记名或字符串表达式。

示例

此字符串返回 0 — 由于这两个字符串被视作完全相同：

```
StringCompareNoCase("Apple","apple")
```

此字符串返回 -6 — 由于这两个字符串被视作不完全相同，第一个不同字符“p”的 ASCII 值减去相应字符“v”的 ASCII 值等于 -6：

```
StringCompareNoCase("Apple","Avocado")
```

StringCompareEncrypted() 函数

比较加密字符串与未加密的字符串，并返回布尔结果。您可以使用此函数来验证口令。如需有关口令加密的详细信息，请参阅 *InTouch® HMI 可视化指南* 中的第 4 章“设置对象动画效果”中的“启用字符串输入”。

语法

```
result = StringCompareEncrypted (plain, encrypted)
```

参数

plain

字符串、消息标记名或字符串表达式。

encrypted

加密的消息标记名。

示例

明文与密文完全相同时，此脚本返回 1；否则返回 0。Passwd 是消息标记，它包含加密的用户输入值。PlainTxt 是要与用户输入作比较的消息标记。

```
StringCompareEncrypted(PlainTxt, Passwd)
```

转换数据类型

在脚本中，通过使用转换 QuickScript 可以将标记名中包含的值转换为其它数据类型。这使您可以用数学函数来处理字符串数据，或是将这些值记录到 ArchestrA® Log Viewer 以便进行调试。

- Text() 函数
- StringFromIntg() 函数
- StringFromReal() 函数
- StringToIntg() 函数
- StringToReal() 函数
- DText() 函数

Text() 函数

Text() 函数根据指定的格式将数字的值当作字符串来返回。您可能希望这样做，以通过某种方式设置某个值的格式，或是将结果同其它字符串值合并起来以便进一步处理。

语法

result = Text (number, format)

参数

number

数值、模拟标记名或数值表达式。

format

使用 “#”、“0”、“.” 或 “,”。

使用 “#” 代表数字、“.” 代表小数点、“0” 代表强制使用前导零、“,” 代表插入逗号。

如果在格式中使用零，则它后面必须跟零。小数点右边所有的位必须总是为零。例如，000.00 正确，而 #0#0.0# 则不正确。

此函数会根据需要对值进行舍入。字符串、消息标记名或字符串表达式。

示例

Text (66, "#.00") 返回 “66.00”。

Text (1234, "#") 返回 “1234”。

Text (123.4, "#,##0.0") 返回 “123.4”。

Text (12.3, "0,000.0") 返回 “0,012.3”。

Text (3.57, "#.#") 返回 “3.6”。

如果模拟标记名 “pressure” 包含值 1690.2743，则此脚本返回字符串 “Reactor Pressure is 1690.3 mbar”。

```
"Reactor Pressure is "+Text(pressure, "#.#")+" mbar"
```

StringFromIntg() 函数

在脚本中，通过使用 `StringFromIntg()` 函数可以将整数值转换为字符串值。

此函数返回整数值字符串值，并同时执行基数转换。举例来说，这可以用于同时显示文本与整数值，或是将整数值转换为十六进制数。

语法

```
result = StringFromIntg (number, base)
```

参数

number

整数值、整型标记名或整型表达式。

base

转换的基数。这用于将值转换到不同的基数，如二进制 (2)、十进制 (10) 或十六进制 (16)。整数值、整型标记名或整型表达式。

示例

`StringFromIntg(26,2)` 返回 “11010” (二进制)。

`StringFromIntg(26,8)` 返回 “32” — 由于 (8 进制: $26 = 3 \times 8 + 2$)

`StringFromIntg(26,10)` 返回 “26” (十进制)。

`StringFromIntg(26,16)` 返回 “1A” (十六进制)。

StringFromReal() 函数

在脚本中，通过使用 `StringFromReal()` 函数可以将实数值转换为字符串值。

您也可以指定：

- 将值舍入到指定的精度。
- 以指数形式传递值。

举例来说，这可以用于同时显示文本与实数值，或用于显示指数形式的实数。

语法

```
result = StringFromReal (number, precision, type)
```

参数

number

值、模拟标记名或数值表达式。

precision

指定要使用的小数位。整数值、整型标记名或整型表达式。

type

指定是否使用指数形式。字符串、消息标记名或字符串表达式。

“f” - 使用浮点记号。

“e” - 使用带小写 “e” 的指数形式。

“E” - 使用带大写 “E” 的指数形式。

示例

```
StringFromReal (263.355, 2, "f") 返回 “263.36”。
```

```
StringFromReal (263.355, 2, "e") 返回 “2.63e2”。
```

```
StringFromReal (263.55, 3, "E") 返回 “2.636E2”。
```

```
StringFromReal (0.5723, 2, "E") 返回 “5.72E-1”。
```

StringToIntg() 函数

在脚本中，通过使用 `StringToIntg()` 函数可以将字符串中包含的值转换为整数值。

您可以用它将字符串开头的值读取到整型标记，以便进行进一步的数学运算。

语法

```
result = StringToIntg (string)
```

参数

string

字符串、消息标记名或字符串表达式。

附注

此函数检查字符串的第一个字符。如果是数字，则它会试图将这个字符以及后续的字符读取到一个整数中，直至遇到非数值字符。此函数忽略字符串中的前导空格。

示例

`StringToIntg("ABCD")` 返回 0。

`StringToIntg("13.4 mbar")` 返回 13。

`StringToIntg("Pressure is 13.4")` 返回 0。

要从字符串 (*mtag*) 中提取不在开头位置的第一个整数并将它存储在整型标记 *itag* 中，请使用以下动作脚本：

```
DIM i AS INTEGER;
DIM tmp AS INTEGER;
FOR i = 1 TO StringLen(mtag)      {run variable i over the characters of mtag}
    tmp = StringASCII(StringMid(mtag, i, 1)) - 48;    {detect ASCII value}
    IF (tmp>=0 AND tmp<10) THEN {if ASCII value represented "0" - "9"}
        itag = StringToIntg(StringMid(mtag, i, 0));    {set itag to value from that
        position and exit loop}
        EXIT FOR;
    ENDIF;
NEXT;
```

StringToReal() 函数

在脚本中，通过使用 **StringToReal()** 函数可以将字符串中包含的值转换为实数值。

您可以用它将字符串开头包含的值读取到某个实型标记，以便进行进一步的数学运算。

备注 此函数也支持指数形式，并可以将 **1e+6** 这样的字符串表达式正确转换为 **1000000**。

语法

result = StringToReal (*string*)

参数

string

字符串、消息标记名或字符串表达式。

附注

此函数检查字符串的第一个字符。如果是数字，则它会将这个字符以及后续的字符读取到某个实数中，直至遇到非数值字符。此函数忽略字符串中的前导空格。

要从字符串（消息标记 **mtag**）中提取不在开头位置的第一个实数并将它存储在实型标记 **rtag1** 中，请使用以下脚本：

```
DIM i AS INTEGER;
DIM tmp AS INTEGER;
FOR i = 1 TO StringLen(mtag)      {run variable i over the characters of mtag}
    tmp = StringASCII(StringMid(mtag, i, 1)) - 48;    {detect ASCII value}
    IF (tmp>=0 AND tmp<10) THEN {if ASCII value represented "0" - "9"}
        rtag = StringToReal(StringMid(mtag, i, 0));    {set rtag to value from that
position and exit loop}
    EXIT FOR;
ENDIF;
NEXT;
```

示例

StringToReal("ABCD") 返回 0。

StringToReal("13.4 mbar") 返回 13.4。

StringToReal("Pressure is 13.4") 返回 0。

DText() 函数

在脚本中，通过使用 **DText()** 函数可以将布尔值转换为字符串值。您可以使用此函数来自定义消息显示动画链接。

此函数根据布尔值的值来返回不同的字符串值。

语法

```
result = Dtext (Boolean, stringtrue, stringfalse)
```

参数

Boolean

布尔值、离散标记名或布尔表达式。

stringtrue

Boolean 为真时要返回的字符串。字符串值、消息标记名或字符串表达式。

stringfalse

Boolean 为假时要返回的字符串。字符串值、消息标记名或字符串表达式。

示例

如果离散标记名 *switch* 为 **TRUE**，此脚本返回 “Running”；否则返回 “Stopped”。

```
DText (switch, "Running", "Stopped")
```

此脚本根据离散标记 *switch1* 的值返回另一个离散标记 *switch2* 的 “打开消息” 与 “关闭消息”。

```
DText (switch1, switch2.OnMsg, switch2.OffMsg)
```


在运行时处理 InTouch 窗口

在脚本中，可以控制 InTouch 窗口的行为与外观。您也可以使用 QuickScripts 编写脚本来打印单独的 InTouch 窗口或整个屏幕。

显示打开的窗口的列表

在脚本中，通过使用 `OpenWindowsList()` 函数可以显示一个对话框，该对话框包含当前打开的 InTouch 窗口的列表。

OpenWindowList() 函数

显示一个对话框，包含当前打开的 InTouch 窗口的列表。

您无法在动画链接中使用此函数。

语法

```
[result = ]OpenWindowsList();
```

示例

此脚本打开 **打开窗口列表** 对话框，并显示当前打开的所有 InTouch 窗口。

```
OpenWindowsList ()
```

检查窗口是打开、关闭或是否存在

在脚本中，通过使用 `WindowState()` 函数可以检查 InTouch 窗口是打开、关闭或是否不存在。

WindowState() 函数

检查某个 InTouch 窗口是打开、关闭或是否不存在。

语法

```
result = WindowState (windowname)
```

参数

windowname

窗口的名称。字符串值、消息标记名或字符串表达式。

返回值

整数值，含义如下：

0 - InTouch 窗口存在，且当前处于关闭状态

1 - InTouch 窗口存在，且当前处于打开状态

2 - InTouch 窗口不存在

示例

如果 InTouch 窗口 *Main* 存在但并未打开，则此脚本返回 0。

```
WindowState ("Main")
```

打开 InTouch 窗口

在脚本中，通过使用以下 QuickScript 函数之一，可以打开某个 InTouch 窗口：

| 函数 | 作用 |
|-----------------|--|
| Show | 在其位置设置所定义的位置上打开 InTouch 窗口。 |
| ShowAt() | 在指定的位置上打开 InTouch 窗口。打开的窗口在该位置处居中显示。此函数也可用于移动打开的窗口。 |
| ShowHome | 打开在 WindowViewer 属性对话框中的 起始窗口 选项卡中所指定的 InTouch 窗口，并关闭任何其它窗口。 |
| ShowTopLeftAt() | 在指定的位置上打开 InTouch 窗口。打开的窗口左上角对齐该位置。此函数也可用于移动打开的窗口。 |

Show() 函数

在缺省位置上打开 InTouch 窗口。

语法

Show *windowname*

参数

windowname

要打开的窗口的名称。字符串值、消息标记名或字符串表达式。

示例

此脚本打开窗口 *Main*。

```
Show "Main";
```

此脚本打开使用 *wname* 消息标记中所存储的名称的窗口。

```
Show wname;
```

ShowAt() 函数

在指定的位置上打开 InTouch 窗口。它也可以将打开的 InTouch 窗口移到指定的位置。该位置是窗口的中点。

备注 如果窗口的一条边在屏幕显示范围之外，则该窗口不会居中。

语法

ShowAt (*windowname*, *xpos*, *ypos*)

参数

windowname

要打开或移动的窗口的名称。

xpos

窗口中心要移到的水平位置，以像素计。值、模拟标记名或数值表达式。

ypos

窗口中心要移到的垂直位置，以像素计。值、模拟标记名或数值表达式。

示例

此脚本打开 *Main* 窗口，使它在 (x:450, y:130) 位置居中。

```
ShowAt ("Main", 450, 130);
```

此脚本打开 *UserDialog* 窗口，并使它在调用此函数（例如，按钮）的对象的位置上进行居中。

```
ShowAt ("UserDialog", $ObjHor, $ObjVer);
```

ShowHome() 函数

打开在 **WindowViewer** 属性对话框**起始窗口**选项卡中所指定的 InTouch 窗口，并关闭任何其它窗口。

语法

```
ShowHome;
```

ShowTopLeftAt() 函数

在指定的位置上打开 InTouch 窗口。也可用于移动打开的窗口。

语法

ShowTopLeftAt (*windowname*, *xpos*, *ypos*)

参数

要打开或移动的窗口的名称。

xpos

窗口左侧边缘要移到的水平位置，以像素计。值、模拟标记名或数值表达式。

ypos

窗口顶部边缘要移到的垂直位置，以像素计。值、模拟标记名或数值表达式。

示例

此脚本打开 *Main* 窗口，并将它的左上角放到 (x:450, y:130) 位置。

```
ShowTopLeftAt ("Main", 450, 130);
```

移动窗口与调整大小

在脚本中，通过使用 `WWMoveWindow()` 函数可以移动打开的 InTouch 窗口或调整其大小。指定的窗口打开时，暂时应用新的位置与新的尺寸。

WWMoveWindow() 函数

将打开的 InTouch 窗口移到指定的位置，并将它调整到指定的大小。指定的窗口打开时，暂时应用新的位置与新的尺寸。

语法

`WWMoveWindow (windowname, xpos, ypos, xsize, ysize)`

参数

windowname

要打开或移动的窗口的名称。

xpos

窗口左侧边缘要移到的水平位置，以像素计。值、模拟标记名或数值表达式。

ypos

窗口顶部边缘要移到的垂直位置，以像素计。值、模拟标记名或数值表达式。

xsize

指定的窗口的水平大小，以像素计。值、模拟标记名或数值表达式。

ysize

指定的窗口的垂直大小，以像素计。值、模拟标记名或数值表达式。

隐藏 InTouch 窗口

在脚本中，通过使用以下函数之一，可以隐藏 InTouch 窗口。

| 函数 | 作用 |
|----------|-----------|
| Hide | 隐藏指定的窗口。 |
| HideSelf | 隐藏当前活动窗口。 |

Hide() 函数

隐藏（关闭）InTouch 窗口。

语法

Hide *windowname*;

参数

windowname

要隐藏的窗口的名称。字符串值、消息标记名或字符串表达式。

示例

此脚本隐藏 *UserConfirmation* 窗口。

```
Hide "UserConfirmation";
```

HideSelf() 函数

隐藏（关闭）当前活动的 InTouch 窗口。

备注 此函数仅能用在动作 QuickScript 中。

语法

HideSelf;

示例

```
HideSelf;
```

更改窗口的颜色

在脚本中，通过使用 `ChangeWindowColor()` 函数可以更改打开的 InTouch 窗口的颜色。

ChangeWindowColor() 函数

更改打开的 InTouch 窗口的颜色并返回结果码。

语法

`Result = ChangeWindowColor (windowname, rValue, gValue, bValue)`

参数

windowname

要更改颜色的窗口的名称。字符串值、消息标记名或字符串表达式。

rValue

红色的浓度。0 到 255 范围内的整数值、整型标记名或整型表达式。

gValue

绿色的浓度。0 到 255 范围内的整数值、整型标记名或整型表达式。

bValue

蓝色的浓度。0 到 255 范围内的整数值、整型标记名或整型表达式。

返回值

含义如下的值：

- 0 - 失败，窗口未定义或 RGB 值超出范围。
- 1 - 成功。
- 2 - 失败。窗口存在，但并未打开。

在运行时打印窗口

在脚本中，通过使用 `PrintWindow()` 或 `PrintScreen()` 函数可以打印单独的 InTouch 窗口或整个 WindowViewer 屏幕。您也可以使用 `SetWindowPrinter()` 函数设置希望使用的打印机。

SetWindowPrinter() 函数

在运行时，可以使用 `SetWindowPrinter()` 函数设置希望使用的打印机。

备注 使用此函数设置的打印机也是 `PrintHT()` 函数所使用的打印机。

语法

`SetWindowPrinter (printername)`

参数

printername

打印机的名称或者显示为网络共享，或者显示为其属性窗口中所显示的打印机名。字符串值、消息标记名或字符串表达式。

示例

在本例中，`PRTSRV1` 是节点名，`PRT22SW1` 是赋予打印机的共享名。

```
SetWindowPrinter("\\PRTSRV1\\PRT22SW1");
```

本例中，`Epson LX-300` 是打印机的名称，如打印机的“属性”窗口中所显示的那样。

```
SetWindowPrinter("Epson LX-300");
```

在本例中，`MyPrinter` 是消息标记，它包含所安装的 Windows 打印机的名称，或是指向共享网络打印机的路径。

```
SetWindowPrinter(MyPrinter);
```


关于打印的建议

以下列表包含打印时应注意的一些事项。对于打印单个窗口或 WindowViewer 屏幕，这些事项都适用。

- 1 在打印之前，打开要打印的窗口。否则“窗口”与“ActiveX 控件”可能不会正确打印。
- 2 您无法打印到当前正在打印报警的打印机。
- 3 打印时避免在窗口上叠放窗口与对象。
- 4 尽可能的使用 True Type 字体。缺省 InTouch 字体 (System) 不是 True Type 字体。
- 5 为了快速打印，请考虑使用白色背景、减少对象，并使用文本代替图形。
- 6 在将窗口发送给打印机队列之前，WindowViewer 会等待一定的时间。在此期间，WindowViewer 会在后台更新该窗口的任何 DDE 值。要更改此等待时间，请打开 intouch.ini 文件，然后更改或添加下面这行（以毫秒计）：
`PrintWindowWait=10000`

PrintWindow() 函数

在脚本中，通过使用 PrintWindow() 函数可以打印 InTouch 窗口。

语法

```
[result = ] PrintWindow (windowname, leftmargin, topmargin, width, height, options);
```

参数

windowname

要打印的窗口的名称。字符串值、消息标记名或字符串表达式。

leftmargin

左侧边距偏移量（以英寸计）。数值、模拟标记名或数值表达式。

topmargin

顶部边距偏移量（以英寸计）。数值、模拟标记名或数值表达式。

width

打印输出宽度（以英寸计）。将此值设置为 0 可得到最大的纵横比。数值、模拟标记名或数值表达式。

height

打印输出高度（以英寸计）。将此值设置为 0 可得到最大的纵横比。数值、模拟标记名或数值表达式。

options

离散值 0 或 1，仅当 *width* 与 *height* 都为 0 时使用。布尔值、离散标记名或布尔表达式。设置为：

- 1 - 窗口按最大纵横比，也就是窗口大小的整数倍打印。
- 0 - 窗口按适合页面的最大纵横比打印。

备注 如果窗口包含位图，则将 *options* 设置为 1 可以防止拉伸位图。

返回值

- 0 - 打印作业未能成功进入队列，或窗口不存在
- 1 - 打印作业成功进入队列

PrintScreen() 函数

您可以使用 PrintScreen() 函数编写脚本来打印整个 WindowViewer 屏幕。

语法

PrintScreen (*ScreenOption*, *PrintOption*)

参数

ScreenOption

确定 WindowViewer 屏幕要打印的大小。整数值、整型标记名或整型表达式。

- 1 - 打印客户端区域，无菜单（缺省值）
- 2 - 打印整个窗口区域，包括菜单

PrintOption

确定如何拉伸打印的图像，使之适合打印输出。

- 1 - 最佳适合：
在不改变纵横比的情况下拉伸图像，使之从水平或垂直方向适合打印输出。（缺省值）
- 2 - 垂直适合：
在不改变纵横比的情况下拉伸图像，使之从垂直方向适合打印输出。可能会水平修剪图像。
- 3 - 水平适合：
在不改变纵横比的情况下拉伸图像，使之从水平方向适合打印输出。可能会垂直修剪图像。
- 4 - 整页缩放：
拉伸图像，使之从水平与垂直方向上都适合打印输出。纵横比可能会更改，但不会截断图像。
- 无效选项，包含 0，缺省为“最佳适合”。

备注 弹出窗口超出 WindowViewer 屏幕之外的区域会被修剪掉。

示例

此脚本将当前的整个 WindowViewer 屏幕区域（不含菜单）的打印输出发送到打印队列。打印输出包含经过拉伸以填满打印输出尺寸的屏幕区域。

```
PrintScreen(1,4);
```

PrintHT() 函数

在脚本中，通过创建按钮并将它链接到执行 PrintHT QuickScript 函数的动作 QuickScript，可以打印历史趋势。

希望打印整个窗口而不只是趋势图表时，请使用 PrintWindow() 函数代替 PrintHT() 函数。

备注 使用打印选项或 PrintHT() 函数打印历史趋势时，不会打印 x 轴与 y 轴的值。要打印 x 轴与 y 轴的值，请使用 PrintWindow() 或 PrintScreen()。

语法

```
PrintHT(HistTrendTagname);
```

参数

HistTrendTagname

要打印的历史趋势的历史趋势标记名。

处理日期与时间信息

在脚本中，通过系统标记与 QuickScript 函数，可以在计算中使用系统时间与日期设置。InTouch 脚本也支持涉及到多个时区与“夏令时”的计算。

检索数值日期与时间信息

在脚本中，通过使用多个数值系统标记与一个脚本函数，可以检索有关系统日期与时间的信息。这些标记与脚本函数可以用在其它数学运算中。系统标记与脚本函数包括：

| 函数 | 作用 |
|---------------|---|
| \$Year | 返回当前年份。 |
| \$Month | 返回当前月份。 |
| \$Day | 返回月份中的当前日期值。 |
| \$Hour | 返回一日中的当前小时值。 |
| \$Minute | 返回当前时刻的分钟值。 |
| \$Second | 返回当前这秒的值。 |
| \$Msec | 返回当前这毫秒的值。 |
| \$Time | 返回自午夜以来的时间（以毫秒为单位，本地时区）。 |
| \$Date | 返回自 1970 年 1 月 1 日以来的整天数（本地时区）。 |
| \$DateTime | 返回自 1970 年 1 月 1 日以来的天数（包括非整天，本地时区）。 |
| DateTimeGMT() | 返回自“通用协调时间”（Coordinated Universal Time, 简称 UTC）1970 年 1 月 1 日以来的天数，包括非整天。 |

\$Year 系统标记

返回当前年份。

语法

\$Year

数据类型

整型（只读）

示例

此脚本将字符串 “Welcome to xxxx” 指定给字符串 *Welcome*, 其中 xxxx 是当前年份。

```
Welcome = "Welcome to " + StringFromIntg($Year,10)
```

\$Month 系统标记

返回当前月份。

语法

\$Month

数据类型

整型（只读）

示例

如果当前月份是 10，此脚本将字符串 “October” 指定给字符串 *MonthName*。

```
IF $Month==10 THEN  
    MonthName="October";  
ENDIF;
```

\$Day 系统标记

返回本月的当前日期值。

语法

\$Day

数据类型

整型（只读）

示例

如果当前日期是 2 月 29 号，则此脚本将字符串 “It is a leap year!” 指定给字符串 *Msg2User*。

```
IF $Day==29 AND $Month==2 THEN  
    Msg2Usr="It is a leap year!";  
ENDIF;
```

\$Hour 系统标记

返回一日中的当前小时值。

语法

\$Hour

数据类型

整型（只读）

示例

此脚本检查是否是下午 8 点并且备份尚未运行（用离散标记 BackupAlreadyRun 表示），如果是，则调用 RunBackup() QuickFunction 脚本并将 BackupAlreadyRun 标帜设为 TRUE。

```
IF $Hour==20 AND BackupAlreadyRun==0 THEN
    CALL RunBackup();
    BackupAlreadyRun=1;
ENDIF;
```

\$Minute 系统标记

返回当前时刻的分钟值。

语法

\$Minute

数据类型

整型（只读）

示例

此脚本检查是否是下午 4:50，如果是，则显示 *Shift End* 窗口。

```
IF $Minute==50 AND $Hour==16 THEN
    Show "Shift End";
ENDIF;
```

\$Second 系统标记

返回当前的秒值。

语法

\$Second

数据类型

整型（只读）

示例

此脚本生成一个振幅为 100、周期为 1 分钟的正弦波函数。

```
100*Sin(6*$Second)
```

此脚本生成一系列每秒钟交替更改的 0 与 1。

```
$second.00
```

\$Msec 系统标记

返回当前的毫秒值。

备注 缺省条件下，InTouch 每 1000 毫秒更新一次所有的标记。因为如此，\$Msec 系统标记似乎并未更改。如果增加 WindowViewer 属性中的更新速率，则可以看见 \$Msec 标记在更新。

语法

\$Msec

数据类型

整型（只读）

\$Time 系统标记

返回当地时间自午夜以来的毫秒数。

语法

\$Time

数据类型

整型（只读）

示例

此脚本返回自午夜以来的秒数。

```
$Time/1000
```


\$Date 系统标记

返回自 1970 年 1 月 1 日以来的整天数。

语法

\$Date

数据类型

整型（只读）

示例

此脚本返回当前时间。

```
StringFromTime(($Date*86400)+($Time/1000),3);
```

\$DateTime 系统标记

返回自 1970 年 1 月 1 日以来的天数，包括非整天。

语法

\$DateTime

数据类型

实型（只读）

示例

此脚本返回当前时间。

```
StringFromTime($DateTime*86400,3);
```

DateTimeGMT() 函数

返回自“通用协调时间”(UTC) 1970 年 1 月 1 日以来的天数，包括非整天。

备注 此函数无法用在动画显示链接中。

语法

```
result = DateTimeGMT();
```

返回值

自 UTC 时间 1970 年 1 月 1 日以来的天数。实数值。

示例

此脚本返回 UTC 格式的当前日期 / 时间。

```
StringFromTime(DateTimeGMT() * 86400.0, 3);
```

检索字符串日期与时间信息

在脚本中，可以检索字符串形式的日期与时间信息。对于在屏幕上显示日期与时间，或是在需要对整个日期 / 时间字符串进行计算时，这非常有用。

您可以使用以下系统标记与脚本函数。

| 函数 | 作用 |
|--------------|-----------------------------|
| \$DateString | 返回短格式的系统日期。 |
| \$TimeString | 返回系统时间。 |
| UTCDateTime | 返回 UTC 日期与 / 或时间以及本地计算机的时区。 |

\$DateString 系统标记

返回本地操作系统“区域设置”中所定义的短格式的系统日期。

语法

\$DateString

数据类型

字符串（只读）

示例

根据操作系统“区域设置”中的短日期格式设置，此脚本可能返回 4/28/2006。

\$DateString

\$TimeString 系统标记

返回本地操作系统“区域设置”中所定义的系统时间。

语法

\$TimeString

数据类型

字符串（只读）

示例

根据操作系统“区域设置”中的时间格式设置，此脚本可能返回 02:40:37 PM。

\$TimeString

UTCDateTime() 函数

返回 UTC 时间、UTC 日期与时间，或本地时区。

语法

result = UTCDateTime (*format*)

参数

format

确定返回的内容。字符串值、消息标记名，或带以下可能值的字符串表达式：

UTC_SHORT - 函数返回 UTC 时间

UTC_LONG - 函数返回 UTC 日期与时间

UTC_LOCAL - 函数返回本地操作系统时区设置中所设置的时区的名称

任何其它值返回缺省格式的 UTC 日期与时间 (ddd mm dd hh:mm:ss yyyy)。

示例

在太平洋时区 2003 年 1 月 6 号星期一上午 09:24，UTCDateTime() 函数返回以下内容。

此脚本返回 17:24:05

```
UTCDateTime("UTC_SHORT")
```

此脚本返回 01/06/2003 17:24:05

```
UTCDateTime("UTC_LONG")
```

此脚本返回 Pacific Standard Time -8:0: 1

```
UTCDateTime("UTC_LOCAL")
```

此脚本返回 Mon Jan 06 17:24:05 2003。

```
UTCDateTime("Invalid")
```

将日期与时间信息转换为字符串

在脚本中，可以将日期与时间信息转换为字符串，以便于解释及满足显示要求。您可以使用以下函数。

| 函数 | 作用 |
|-----------------------|--------------------------------|
| StringFromTime() | 将 UTC 时间标签转换为本地时间并作为时间字符串返回。 |
| wwStringFromTime() | 将本地时间标签转换为 UTC 时间并将它作为时间字符串返回。 |
| StringFromTimeLocal() | 将时间标签转换为时间字符串。 |

StringFromTime() 函数

将 UTC 时间格式的时间标签转换为本地时间并返回作为字符串的结果。此函数支持“夏令时”。

备注 此函数等同于 StringFromGMTTimeToLocal() 函数。

语法

result = StringFromTime (timestamp, format)

参数

timestamp

自 UTC 时区 1970 年 1 月 1 日午夜以来经过的秒数。整数值、整型标记名或整型表达式。

format

确定如何显示字符串结果。1 到 5 范围内的整数值、整型标记名或整型表达式，含义如下：

- 1 - 根据本地操作系统“区域设置”中所设置的格式显示日期
- 2 - 根据本地操作系统“区域设置”中所设置的格式显示时间
- 3 - 按 24 个字符的字符串格式显示日期与时间 (ddd mmm dd hh:mm:ss yyyy)
- 4 - 按短格式显示星期几
- 5 - 按长格式显示星期几

示例

本例假设本地节点上的时区是“太平洋标准时间”(PST, UTC-0800)。传递给函数的 UTC 时间是 1970 年 1 月 2 日星期五上午 12 点。因为 PST 比 UTC 晚 8 小时，函数返回以下结果。

此脚本返回“1/1/70”

```
StringFromTime(86400,1)
```

此脚本返回 “04:00:00 PM”

```
StringFromTime(86400,2)
```

此脚本返回 “Thu Jan 01 16:00:00 1970”

```
StringFromTime(86400,3)
```

此脚本返回 “Thu”

```
StringFromTime(86400,4)
```

此脚本返回 “Thursday”

```
StringFromTime(86400,5)
```

wwStringFromTime() 函数

将本地时间格式的时间标签转换为 UTC 时间并返回作为字符串的结果。此函数支持 “夏令时”。

语法

```
result = wwStringFromTime (timestamp, format)
```

参数

timestamp

自本地时区 1970 年 1 月 1 日午夜以来经过的秒数。整数值、整型标记名或整型表达式。

format

确定如何显示字符串结果。1 到 5 范围内的整数值、整型标记名或整型表达式，含义如下：

- 1 - 根据本地操作系统 “区域设置” 中所设置的格式显示日期
- 2 - 根据本地操作系统 “区域设置” 中所设置的格式显示时间
- 3 - 按 24 个字符的字符串格式显示日期与时间 (ddd mmm dd hh:mm:ss yyyy)
- 4 - 按短格式显示星期几
- 5 - 按长格式显示星期几

示例

本例假设本地节点上的时区是 “太平洋标准时间” (PST, UTC-0800)。传递给函数的本地时间是 1970 年 1 月 1 日星期四下午 04:00:00。因为 PST 比 UTC 晚 8 小时，函数返回以下结果。

此脚本返回 “1/2/70”

```
wwStringFromTime(57600,1)
```

此脚本返回 “12:00:00 AM”

```
wwStringFromTime(57600,2)
```

此脚本返回 “Fri Jan 02 00:00:00 1970”

```
wwStringFromTime(57600,3)
```

此脚本返回 “Fri”

```
wwStringFromTime(57600,4)
```

此脚本返回 “Friday”

```
wwStringFromTime(57600,5)
```

StringFromTimeLocal() 函数

将时间标签转换为时间并返回作为字符串的结果。

语法

```
result = StringFromTimeLocal (timestamp, format)
```

参数

timestamp

自 1970 年 1 月 1 日午夜以来经过的秒数。整数值、整型标记名或整型表达式。

format

确定如何显示字符串结果。1 到 5 范围内的整数值、整型标记名或整型表达式，含义如下：

- 1 - 根据本地操作系统“区域设置”中所设置的格式显示日期
- 2 - 根据本地操作系统“区域设置”中所设置的格式显示时间
- 3 - 按 24 个字符的字符串格式显示日期与时间 (ddd mmm dd hh:mm:ss yyyy)
- 4 - 按短格式显示星期几
- 5 - 按长格式显示星期几

示例

此脚本返回 “1/2/70”

```
StringFromTimeLocal(86400,1)
```

此脚本返回 “12:00:00 AM”

```
StringFromTimeLocal(86400,2)
```

此脚本返回 “Fri Jan 02 00:00:00 1970”

```
StringFromTimeLocal(86400,3)
```

此脚本返回 “Fri”

```
StringFromTimeLocal(86400,4)
```

此脚本返回 “Friday”

```
StringFromTimeLocal(86400,5)
```

检查夏令时状态

在脚本中，通过使用 `wwIsDaylightSaving()` 函数可以检查是否在使用夏令时。

`wwIsDaylightSaving()` 函数

返回当前是否在使用夏令时。

语法

```
result = wwIsDaylightSaving()
```

返回值

含义如下的布尔值：

- 0 - “夏令时” 未在使用。
- 1 - “夏令时” 正在使用。

与其它应用程序交互

在脚本中，通过使用各种 QuickScript 可以与其它 Windows 应用程序进行交互。例如，可以：

- 启动应用程序，如“记事本”。
- 选择应用程序标题名。
- 检查某个应用程序是否正在运行。
- 激活正在运行的应用程序。
- 模拟键击。
- 关闭、最小化或最大化应用程序窗口。
- 执行命令并与支持 DDE 的应用程序交换数据。

启动 Windows 应用程序

在脚本中，可以使用 StartApp 命令启动 Windows 应用程序。

语法

StartApp appname;

参数

appname

要启动的应用程序的路径与文件名。字符串值、消息标记名或字符串表达式。

备注 您需要知道应用程序的路径与文件名。如果应用程序所在的目录是 Windows PATH 环境变量的一部分，则只要传递文件名（不需要路径）。

示例

此脚本启动 Microsoft 的“计算器”。

```
StartApp "calc"
```


检索正在运行的应用程序的应用程序标题

在脚本中，通过使用 `InfoAppTitle()` 函数，可以找到指定的正在运行的应用程序的应用程序标题或 Windows 任务列表名。举例来说，可以通过 `InTouch` 脚本来获取此信息，然后利用此信息来检查指定的应用程序当前是否正在运行，或用来激活它。

InfoAppTitle() 函数

返回指定的正在运行的应用程序的应用程序标题或 Windows 的任务列表名。

语法

```
result = InfoAppTitle (appname)
```

参数

appname

不带 .exe 扩展名的应用程序名。字符串值、消息标记名或字符串表达式。

示例

此脚本返回 “计算器”

```
InfoAppTitle ("calc")
```

此脚本返回 “Microsoft Excel”

```
InfoAppTitle ("excel")
```

检查应用程序是否正在运行

在脚本中，通过使用 `InfoAppActive()` 函数可以检查指定的应用程序是否已在运行。您需要先知道应用程序标题或 Windows 任务列表名，然后才可以检查特定的应用程序是否正在运行。

InfoAppActive() 函数

返回应用程序的运行状态。

语法

```
result = InfoAppActive (apptitle)
```

参数

apptitle

希望查询其运行状态的应用程序的应用程序标题或 Windows 任务列表名。字符串值、消息标记名或字符串表达式。

返回值

含义如下的布尔值：

0 - 应用程序未在运行

1 - 应用程序正在运行

示例

此脚本查询“记事本”应用程序；如果已在运行，则激活它。否则该脚本启动新的“记事本”实例。这就避免了多次启动“记事本”。

```
IF InfoAppActive(InfoAppTitle("Notepad"))==1
THEN
    ActivateApp InfoAppTitle( "Notepad" );
ELSE
    StartApp "Notepad";
ENDIF;
```

激活正在运行的 Windows 应用程序

在脚本中，通过使用 `ActivateApp()` 函数，可以激活正在运行的 Windows 应用程序。这会将指定的应用程序带到前台并将焦点切换到它上面。

激活正在运行的 Windows 应用程序之前，需要执行以下操作：

- 查找应用程序标题或 Windows 任务列表名。请参阅第 121 页的“检索正在运行的应用程序的应用程序标题”。
- 确保 Windows 应用程序正在运行。请参阅第 121 页的“检查应用程序是否正在运行”。

ActivateApp 函数

激活已在运行的 Windows 应用程序。

语法

```
ActivateApp apptitle;
```

参数

apptitle

希望激活的正在运行的应用程序的应用程序标题或 Windows 任务列表名。

示例

此脚本检查命令提示符窗口是否已打开；如果是，则激活它。否则启动命令提示符窗口。

```
IF InfoAppActive( InfoAppTitle("cmd")) == 1 THEN
    ActivateApp InfoAppTitle("cmd");
ELSE
    StartApp "cmd";
ENDIF;
```

将模拟键击发送到应用程序

在脚本中，可以模拟键盘上的按键序列。例如，您可以使用它来：

- 在打开的应用程序中自动输入数据
- 控制任何应用程序（包括 InTouch HMI）。

SendKeys 函数

模拟键击序列。

语法

SendKeys *sequence*;

参数

sequence

要模拟的键击序列。字符串值、消息标记名或字符串表达式。

除键盘上的常规字符（如字母数字字符）之外，也可以使用代码来指定控制键：

```
{BACKSPACE} - 模拟 Backspace 键
{BREAK} - 模拟 Break 键
{CAPSLOCK} - 模拟 Caps Lock 键
{DELETE} - 模拟 Delete 键（或 {DEL}）
{DOWN} - 模拟“下箭头”键
{END} - 模拟 End 键
{ENTER} - 模拟 Enter 键（或 ~）
{ESCAPE} - 模拟 ESC 键（或 {ESC}）
{F1} .. {F12} - 模拟 F1 .. F12 键
{HOME} - 模拟 Home 键
{INSERT} - 模拟 Insert 键
{LEFT} - 模拟“左箭头”键
{NUMLOCK} - 模拟 Num Lock 键
{PGDN} - 模拟 Page Down 键
{PGUP} - 模拟 Page Up 键
{PRTSC} - 模拟 Print Screen 键
{RIGHT} - 模拟“右箭头”键
{TAB} - 模拟 Tab 键
{UP} - 模拟“上箭头”键

+ - 模拟 Shift 键
    同要和 Shift 键构成组合键并用括号括起来的键一起使用。

^ - 模拟 Ctrl 键
    同要和 Ctrl 键构成组合键并用括号括起来的键一起使用。

% - 模拟 Alt 键
    同要和 Alt 键构成组合键并用括号括起来的键一起使用。
```

附注

使用 StartApp 与 / 或 ActivateApp() 命令，在将模拟的键击发送到另一个应用程序之前将它激活。

示例

此脚本模拟按 B 键。

```
SendKeys "b";
```

此脚本模拟按 **Ctrl** 与 **P** 组合键，这可用于在另一个应用程序中启动“打印”对话框。

```
SendKeys "^ (p)";
```

此脚本模拟按 **F1**（可以打开帮助功能），按 **Tab** 键（可以将光标放入搜索字段），输入 **HAL**，然后按 **Enter** 键（可以启动搜索）。

```
SendKeys "{F1}{TAB}HAL{ENTER}";
```

此脚本模拟按 **Ctrl**、**Shift**、**1** 组合键，这与切换到 **WindowMaker** 效果相同。这项强大的组合可以用于开发可自我修改的（动态的）**InTouch HMI** 应用程序。

```
SendKeys "^ (+ (1) )";
```

关闭、最小化或最大化 Windows 应用程序

在脚本中，通过使用 `WWControl()` 命令，可以关闭、最小化或最大化其它 Windows 应用程序。

在关闭、最小化或最大化 Windows 应用程序之前，需要执行以下操作：

- 查找应用程序标题或 Windows 任务列表名。请参阅第 121 页的“检索正在运行的应用程序的应用程序标题”。
- 确保该 Windows 应用程序正在运行。请参阅第 121 页的“检查应用程序是否正在运行”。

WWControl() 函数

还原、最小化、最大化或关闭 Windows 应用程序。

语法

`WWControl (apptitle, control);`

参数

apptitle

希望还原、最小化、最大化或关闭的正在运行的应用程序的应用程序标题或 Windows 任务列表名。字符串值、消息标记名或字符串表达式。

control

确定要对指定的 Windows 应用程序执行的操作。使用以下值的字符串值、消息标记名或字符串表达式：

Restore - 激活并显示应用程序窗口

Minimize - 激活并最小化应用程序窗口

Maximize - 激活并最大化应用程序窗口

Close - 关闭应用程序

附注

要在 Windows Server 2003 中使用此函数，必须是本地计算机上 Administrators（管理员）组、Performance Log Users（性能日志用户）组或 Performance Monitor Users（性能监视用户）组的成员，或必须是已被授予正确的注册表写入权限的用户。

示例

如果计算器应用程序已在运行，此脚本可以将它还原。

```
WWControl ("Calculator","Restore");
```

此脚本关闭 WindowViewer。

```
WWControl (InfoAppTitle("View"),"Close");
```

使用 DDE 执行命令与交换数据

您可以编写一个脚本，同支持 DDE 的应用程序进行交互。

| 函数 | 作用 |
|-------------|---------------|
| WWExecute() | 发送与执行命令。 |
| WWRequest() | 从 DDE 项目读取数据。 |
| WWPoke() | 向 DDE 项目写入数据。 |

WWExecute() 函数

向应用程序发送命令、执行它并返回状态结果。您可以使用它来让 Excel 运行某个宏。

语法

Result = WWExecute (*appname*, *topic*, *command*)

参数

appname

命令所发送到的应用程序的名称。字符串值、消息标记名或字符串表达式。

topic

命令所发送到的应用程序内的主题的名称。字符串值、消息标记名或字符串表达式。

command

要发送的命令。字符串值、消息标记名或字符串表达式。

返回值

含义如下的值 -1、0 或 1:

- 1 - 命令未成功执行。可能的原因是应用程序未在运行、主题不存在或命令有错误。
- 0 - 应用程序正忙，导致命令未成功执行。
- 1 - 命令成功执行。

示例

此脚本通过向 Excel 发送命令 `[Run("Macro1" ,0)]` 指示 Microsoft Excel 来执行 *Macro1* 这个宏。

```
Macro="Macro1";
Command="[Run(" + StringChar(34) + Macro + StringChar(34) + ",0)"]";
WWExecute("excel","system",Command);
```

WWRequest() 函数

从应用程序的项目中读取数据。例如，您可以使用它来读取 Microsoft Excel 电子表格单元格的值。

语法

Result = WWRequest(*appname*, *topic*, *item*, *messagetag*)

参数

appname

应用程序的名称。字符串值、消息标记名或字符串表达式。

topic

应用程序中主题的名称。字符串值、消息标记名或字符串表达式。

item

属于该主题与应用程序的项目的名称。字符串值、消息标记名或字符串表达式。

messagetag

检索项目值的消息标记名。通过使用 **StringToIntg()** 或 **StringToReal()** 函数，可以将消息标记名值转换成整数或实数值。

返回值

含义如下的值 -1、0 或 1：

- 1 - 数据未成功读取。可能的原因是应用程序未在运行，或是主题或项目不存在。
- 0 - 应用程序正忙，导致数据未成功读取。
- 1 - 数据读取成功。

示例

此脚本将 Microsoft Excel 工作簿 *Book1.xls* 工作表 *Sheet1* 第 1 行第 1 列中的值读取到消息标记名 *MTag*，并将值放入实型标记名 *CellValue* 中。

```
Result = WWRequest("excel","[Book1.xls]sheet1", "r1c1",Mtag);
CellValue=StringToReal(MTag);
```

如果使用非英文操作系统，则在将 *MTag* 转换为不同的数据类型之前，可能需要使用 **StringReplace()** 函数来更改其内容。例如，对于使用英文逗号作为小数点的操作系统，在将 *MTag* 转换为实型数据类型之前，可能需要使用小数点替换其中的所有逗号。

WWPoke() 函数

向应用程序的项目写入数据。例如，您可以使用它将值写入 Microsoft Excel 中的电子表格单元格。

语法

```
result = WWPoke (appname, topic, item, string)
```

参数

appname

应用程序的名称。字符串值、消息标记名或字符串表达式。

topic

应用程序中主题的名称。字符串值、消息标记名或字符串表达式。

item

属于该主题与应用程序的项目名。字符串值、消息标记名或字符串表达式。

string

要写入的值。字符串值、消息标记名或字符串表达式。通过使用 `StringFromIntg()`、`StringFromReal()` 或 `Text()` 函数，可以将整数或实型标记名的值转换为消息标记名。

返回值

含义如下的值 -1、0 或 1:

- 1 - 数据未成功写入。可能的原因是应用程序未在运行，或是主题或项目不存在。
- 0 - 应用程序正忙，导致数据未成功写入。
- 1 - 数据写入成功。

附注

请勿使用 `WWPoke()` 或 `WWRequest()` 函数在不同节点或会话上的 InTouch 应用程序之间读取和写入数据。要在 InTouch 应用程序之间读取和写入数据，请使用“访问名”来替代。请参阅 *InTouch® HMI 数据管理指南* 中的第 5 章“使用 I/O 进行数据访问”中的“设置访问名”。

示例

此脚本将实型标记名 *CellValue* 的值放入消息标记名 *Mtag* 中，并将该值写入 Microsoft Excel 工作簿 *Book1.xls* 中工作表 *Sheet1* 的第 1 行第 1 列这个电子表格单元格。

```
MTag = Text(CellValue,"0");
Result = WWPoke("excel","[Book1.xls]sheet1", "r1c1",Mtag);
```


处理文件

您可以使用各种文件管理与访问操作来编写脚本。

| 函数 | 作用 |
|--|-----------------|
| FileCopy() | 复制文件。 |
| FileDelete() | 删除文件。 |
| FileMove() | 移动文件。 |
| FileReadFields()、 FileWriteFields() | 读取 / 写入 csv 数据。 |
| FileReadMessage()、 FileWriteMessage() | 读取 / 写入文本数据。 |

管理文件

在脚本中，可以复制、删除或移动文件。

FileCopy() 函数

将源文件复制到目标文件，并返回状态结果。此函数可能要花费较长时间并分为多个阶段来执行：

- 1 调用 FileCopy() 函数并返回中间结果，指出文件复制初始化操作是成功还是失败。
- 2 FileCopy() 函数在后台执行复制过程，InTouch 脚本在复制文件的过程中继续执行。您可以使用一个整型标记来监视文件复制进度。
- 3 FileCopy() 函数返回文件复制结果，指出文件复制过程是成功还是失败。

如果目标文件夹不可用（例如，在网络中的另一台计算机上），则该函数最多等待 10 秒便会超时，然后向 Logger 发送一条消息。

备注 请勿在异步 QuickFunction 中使用 FileCopy() 函数。

语法

```
result = FileCopy (sourcefile, destfile, progresstag)
```

参数

sourcefile

要复制的文件的完整路径与文件名。字符串值、消息标记名或字符串表达式。您可以在该参数中使用通配符（* 与 ?）只复制与指定的准则匹配的文件。路径名也可以是 UNC 路径名。

destfile

目标文件的完整路径与文件名（或仅为路径名）。字符串值、消息标记名或字符串表达式。路径名也可以是 UNC 路径。

progresstag

括在英文双引号中的整型标记的名称，该标记将包含指示文件复制进度的值。字符串值、消息标记名（如包含“IntTag.Name”值的消息标记）或字符串表达式。这些值的含义如下：

- 0 - FileCopy() 过程仍在进行。
- 1 - FileCopy() 过程成功完成。
- 1 - FileCopy() 过程已完成，但发生了错误。

返回值

含义如下的值 -1、0 或 1：

- 1 - FileCopy() 函数调用成功。
- 0 - 调用 FileCopy() 函数时发生错误，原因是另一个 FileCopy() 过程已经在进行。
- 1 - 调用 FileCopy() 函数时发生错误，原因是源文件不存在或目标文件为只读。

示例

此脚本将 c:\MyData\output.log 文件复制到 d:\archive 目录，并将该文件重命名为 output.txt。文件复制的进度写入整型标记 *Monitor*。

```
Status=FileCopy("c:\MyData\output.log","d:\archive\output.txt","Monitor");
```

此脚本将 c:\root 目录中以 .txt 结尾的所有文件复制到目标目录 c:\Backup。

```
Status=FileCopy("c:\*.txt", "c:\Backup", "Monitor");
```

此脚本将完整的路径与文件名包含在消息标记 LogFile 中的文件复制到目标目录 c:\results\，并将它重命名为 logxxx.txt，其中 xxx 是时间标签。

```
Status=FileCopy(LogFile, "c:\results\log" + $DateString + $TimeString + ".txt", "Monitor");
```

FileDelete() 函数

删除单独的文件。

语法

```
result = FileDelete (filename)
```

参数

filename

要删除的文件的路径名与文件名。字符串值、消息标记名或字符串表达式。支持 UNC 路径名。

附注

请勿在 FileDelete() 函数中使用通配符 (* 与 ?)，也不要异步 QuickFunction 中使用 FileDelete() 函数。

FileDelete() 函数不删除目录。

返回值

指出文件删除操作是成功还是失败的值：

- 1 - 文件成功删除
- 0 - 文件删除失败。可能的原因是试图删除只读或不存在的文件。

示例

此脚本删除文件 c:\Data.txt，如果找到该文件并成功删除则返回 1。

```
Status=FileDelete("c:\Data.txt");
```

FileMove() 函数

将源文件移到目标文件并返回状态结果。它也可以用于重命名文件。此函数可能要花费较长时间并分为多个阶段来执行：

- 1 调用 FileMove() 函数并返回中间结果，指出文件移动初始化操作是成功还是失败。
- 2 FileMove() 函数在后台执行移动过程，InTouch 脚本在文件移动的过程中继续执行。您可以使用整型标记来监视文件移动进度。
- 3 FileMove() 函数返回文件移动结果，指出文件移动过程是成功还是失败。

请勿在异步 QuickFunctions 中使用 FileMove() 函数。

语法

```
result = FileMove (sourcefile, destfile, progresstag)
```

参数

sourcefile

要移动的文件完整路径与文件名。字符串值、消息标记名或字符串表达式。您可以在该参数中使用通配符（* 与 ?）只移动与指定的准则匹配的文件。路径名也可以是 UNC 路径名。

destfile

目标文件的完整路径与文件名（或仅为路径名）。字符串值、消息标记名或字符串表达式。路径名也可以是 UNC 路径。

progresstag

括在英文双引号中的整型标记的名称，该标记将包含指出文件移动进度的值。字符串值、消息标记名（例如，包含“IntTag”值的消息标记名）或字符串表达式。这些值的含义如下：

- 0 - FileMove() 过程仍在进行
- 1 - FileMove() 过程成功完成
- 1 - FileMove() 过程已完成，但发生了错误

返回值

含义如下的值 -1、0 或 1：

- 1 - FileMove() 函数调用成功
- 0 - 调用 FileMove() 函数时发生错误，原因是另一个 FileMove() 过程已经在进行
- 1 - 调用 FileMove() 函数时发生错误。可能的错误是试图移动不存在的文件。

示例

此脚本将 c:\MyData\output.log 文件移到 d:\archive 目录，并将该文件重命名为 output.txt。文件移动的进度写入整型标记 *Monitor*。

```
Status=FileMove("c:\MyData\output.log","d:\archive\output.txt","Monitor");
```

此脚本将 c:\root 目录中以 .txt 结尾的所有文件移到目标目录 c:\Backup。

```
Status=FileMove("c:\*.txt", "c:\Backup", "Monitor");
```

此脚本将完整的路径与文件名包含在消息标记 LogFile 中的文件移到目标目录 c:\results\，并将它重命名为 logxxx.txt，其中 xxx 是时间标签。

```
Status=FileMove(LogFile, "c:\results\log" + $DateString + $TimeString + ".txt", "Monitor");
```

读取和写入 CSV 数据

通过使用 FileReadFields() 与 FileWriteFields() 函数，可以编写脚本将 csv（逗号分隔变量）文件中包含的数据读取到一系列的标记名中，也可以将一系列标记名中的数据写入 csv 文件。

FileReadFields() 与 FileWriteFields() 函数仅支持逗号作为分隔符。

FileReadFields() 函数

将 csv 文件中包含的值读取到一系列的标记名中。您可以使用此函数来加载一组标记名值。

逗号是支持的唯一分隔符。

语法

```
[result = ] FileReadFields (filename, offset, starttag, numberoffields)
```

参数

filename

要从中读取数据的 csv 文件的名称。字符串值、消息标记名或字符串表达式。

offset

文件中要开始读取的位置（以字节计）。整数值、整型标记名或整型表达式。

starttag

接收第一个读取数据项的第一个标记名的名称。标记名必须用英文双引号括起且以数字结尾，如 “MyTag1”。字符串值、消息标记名（例如，包含 “MyTag1” 值的消息标记名）或字符串表达式。

numberoffields

要从 csv 文件中读取的数据项的数量。整数值、整型标记名或整型表达式。第一个数据项读取到 starttag 参数定义的标记名，后续的数据项读取到编号为 starttag 参数后缀的增量的标记名（MyTag1、MyTag2、MyTag3...）。

返回值

可选的新文件读取数据之后的偏移量（以字节计）。这可以用于读取下一组数据。

示例

如果 csv 文件 c:\set.csv 包含以下数据：Flour、27.23、14、1，且定义了以下标记：RecipeTag1:message、RecipeTag2:real、Recipe3:integer、RecipeTag4:discrete，则此脚本将“Flour”值读取到 RecipeTag1、将 27.23 读取到 RecipeTag2、将 14 读取到 RecipeTag3、将 1 读取到 RecipeTag4，并返回新文件偏移量。

```
FileReadFields("c:\set.csv",0,"RecipeTag1",4);
```

FileWriteFields() 函数

将一系列标记名中包含的值写入 csv 文件。您可以使用此函数保存一组标记名值。

逗号是支持的唯一分隔符。

语法

[result =] FileWriteFields (filename, offset, starttag, numberoffields)

参数

filename

要写入数据的 csv 文件的名称。如果先前不存在，则创建新文件。字符串值、消息标记名或字符串表达式。

offset

文件中要开始写入的位置（以字节计）。使用 -1 可写入文件末尾（附加）。整数值、整型标记名或整型表达式。

starttag

包含要写入的第一个数据项的第一个标记名的名称。标记名必须用英文双引号括起且以数字结尾，如 “MyTag1”。字符串值、消息标记名（例如，包含 “MyTag1” 值的消息标记名）或字符串表达式。

numberoffields

要写入 csv 文件的数据项的数量。整数值、整型标记名或整型表达式。第一个数据项从 starttag 参数定义的标记名写入文件，后续的数据项从后缀为 starttag 参数的增量的标记名（MyTag1、MyTag2、MyTag3...）写入。

返回值

可选的新文件写入数据之后的偏移量（以字节计）。这可以用于写入下一组数据。

示例

一系列 InTouch 标记定义如下：

| 标记名 | 数据类型 | 值 |
|------------|------|-------|
| RecipeTag1 | 消息 | Flour |
| RecipeTag2 | 实型 | 27.23 |
| RecipeTag3 | 整型 | 14 |
| RecipeTag4 | 离散 | 1 |

此脚本将 RecipeTag1 到 RecipeTag4 中包含的值写入 csv 文件 c:\set.csv。

FileWriteMessage("c:\set.csv",0,"RecipeTag1",4);

使文件 c:\set.csv 包含以下数据： Flour,27.23,14,1

读取和写入文本数据

通过使用 `FileReadMessage()` 与 `FileWriteMessage()` 函数，可以编写脚本从文件中读取文本数据，以及将文本数据写入文件。您可以读取 / 写入指定数量的字节，也可以是整行的文本（用换行符分割）。

`FileReadMessage()` 函数

从文件中读取指定字节数（或一行）的字符串数据。

语法

```
[result = ] FileReadMessage (filename, offset, messagetag, charstoread)
```

参数

filename

要从中读取数据的文件的名称。字符串值、消息标记名或字符串表达式。

offset

文件中要开始读取的位置（以字节计）。整数值、整型标记名或整型表达式。

messagetag

从文件接收第一行或一定字节数的消息标记名。

charstoread

从文件读取的字节数。将它设置为 0 可以一直读取到下一个换行符 (LF)。整数值、整型标记名或整型表达式。

返回值

包含读取之后的新字节位置。您可以用它来从文件中继续进行读取。

示例

此脚本将 `c:\Data\File.txt` 文件中的第一行数据读取到消息标记名 `MsgTag`。

```
FileReadMessage ("c:\Data\File.txt",0,MsgTag, 0);
```


FileWriteMessage() 函数

将指定字节数（或一行）的字符串数据写入文件。

语法

```
[result = ] FileWriteMessage (filename, offset, messagetag, linefeed)
```

参数

filename

要写入数据的文件的名称。字符串值、消息标记名或字符串表达式。

offset

文件中要开始写入的位置（以字节计）。将它设置为 -1 可以将数据写入文件的末尾（附加）。整数值、整型标记名或整型表达式。

messagetag

包含要写入文件的数据的消息标记名。

linefeed

指定在将数据写入文件之后是否写入换行符 (LF)。设置为 1 写入换行符；否则设置为 0。布尔值、离散标记名或布尔表达式。

返回值

包含写入之后的新字节位置。您可以用它来继续写入文件。

示例

此脚本将消息标记名 *MsgTag* 的值写入 c:\Data\File.txt 文件的末尾。

```
FileWriteMessage("c:\Data\File.txt",-1,MsgTag,1);
```

检索系统相关信息

在脚本中，可以使用以下 QuickFunction 来检索系统相关信息。

| 函数 | 作用 |
|-----------------|---------------------|
| GetNodeName() | 检索计算机的节点名。 |
| InfoDisk() | 检索磁盘空间信息。 |
| InfoFile() | 检索文件的有关信息。 |
| InfoResources() | 检索 Windows 环境的有关信息。 |

检索计算机的节点名。

在脚本中，通过使用 GetNodeName() 函数可以检索计算机的节点名。例如，在处理访问名时，这可以用于使 InTouch 应用程序保持动态。

GetNodeName() 函数

返回计算机的节点名。

语法

```
GetNodeName (messagetag, nodenum);
```

参数

messagetag

将包含节点名的消息标记名。

nodenum

要从节点名检索的字符数。0 到 131 范围内的整数值、整型标记名或整型表达式。

示例

此脚本检索节点名，并将它指定给 NodeName 消息标记名。

```
GetNodeName(NodeName,131);
```

检索磁盘空间信息

在脚本中，通过使用 `InfoDisk()` 函数可以检索磁盘空间信息。您可以检索：

- 磁盘驱动器的总计大小（以字节或千字节计）。
- 磁盘驱动器的可用空间（以字节或千字节计）。

通过指定触发器标记，也可以确定在什么时间、按什么频率来更新信息（在动画链接中）。

InfoDisk() 函数

返回本地或网络磁盘驱动器的总计空间或可用空间。

语法

```
result = InfoDisk (drive, infotype, trigger);
```

参数

drive

要检索其信息的盘符。仅使用字符串的第一个字符。字符串值、消息标记名或字符串表达式。

infotype

指定信息类型。使用以下可能值的整数值、整型标记名或整型表达式：

- 1 - 函数返回磁盘驱动器的总计大小（以字节计）
- 2 - 函数返回磁盘驱动器的可用空间（以字节计）
- 3 - 函数返回磁盘驱动器的总计大小（以千字节计）
- 4 - 函数返回磁盘驱动器的可用空间（以千字节计）

trigger

用作重新计算磁盘信息的触发器的标记名（或表达式）。如果触发器值发生改变，则重新计算磁盘信息。离散或模拟标记名，或是离散或模拟表达式。

附注

触发器标记仅当 `InfoDisk()` 函数用在动画显示链接中时才有意义。如果此函数用在脚本中，则可以指定任何数值、模拟型标记名或数值表达式。

示例

在动画显示链接中使用此脚本显示磁盘驱动器 C 的可用空间，并且每分钟更新一次信息。

```
InfoDisk("C", 4, $Minute)
```

检索文件或目录的有关信息

在脚本中，通过使用 `InfoFile()` 函数，可以检索特定文件或目录的有关信息。通过使用不同的参数，可以查找：

- 文件是否存在。
- 指定的文件名实际上是不是一个目录。
- 文件大小（以字节计）。
- 文件或目录的时间标签。
- 与通配符搜索匹配的文件数。

InfoFile() 函数

返回文件或目录的各种有关信息。

语法

```
result = InfoFile (filename, infotype, trigger)
```

参数

filename

要检索相关信息的文件或目录的完整文件名或目录名。字符串值、消息标记名或字符串表达式。也可以包含通配符，如“*”与“?”。

infotype

您希望检索的关于指定文件或目录的信息类型。使用以下可能值且含义如下的整数值、整型标记名或整型表达式：

- 1 - 存在。如果文件存在，`InfoFile()` 函数返回 1；如果文件是目录，返回 2；如果文件或目录不存在，返回 0。
- 2 - 大小。`InfoFile()` 函数返回文件大小（以字节计）。
- 3 - 创建时间标签。`InfoFile()` 函数返回时间标签（使用自 1970 年 1 月 1 日午夜以来经过的秒数表示）。使用 `StringFromTimeLocal()` 函数将这个值转换为消息时间标签。
- 4 - 通配符搜索匹配。`InfoFile()` 函数返回同指定的通配符搜索相匹配的文件数。

trigger

用作重新计算文件信息的触发器的标记名（或表达式）。如果触发器值发生改变，则重新计算文件信息。离散或模拟标记名，或是离散或模拟表达式。

附注

触发器标记仅当 `InfoFile()` 函数用在动画显示链接中时才有意义。如果此函数用在脚本中，则可以指定任何数值、模拟型标记名或数值表达式。

示例

此脚本在 c:\data\log.txt 文件存在时返回 1。

```
InfoFile("c:\data\log.txt",1,$minute)
```

如果 c:\data\log.txt 文件的大小为 14223 字节，此脚本返回 14223。

```
InfoFile("c:\data\log.txt",2,$minute)
```

如果 c:\data\log.txt 文件是在 2006 年 1 月 26 日上午 11:14:26 创建的，此脚本返回 1138245266。

```
InfoFile("c:\data\log.txt",3,$minute)
```

如果 c:\data\ 目录中以 txt 结尾的文件有 14 个，此脚本返回 14。

```
InfoFile("c:\data\*.txt",4,$minute)
```

检索 Windows 环境有关的信息

在脚本中，通过使用 InfoResources() 函数可以检索 Windows 环境的有关信息。您可以查找：

- 页面文件的可用字节。
- Windows 任务的大致数量。

InfoResources() 函数

返回页面文件的可用字节数，或 Windows 任务的大致数量。

语法

```
result = InfoResources (infotype, trigger)
```

参数*infotype*

要检索的 Windows 环境相关信息的类型。使用以下可能值且含义如下的整数值、整型标记名或整型表达式：

- 1 - 页面文件的可用字节数。
- 2 - 打开的 Windows 任务的大致数量。这可以用于衡量系统负载。

trigger

用作检索系统信息的触发器的标记名（或表达式）。如果触发器值发生改变，则再次检索系统信息。离散或模拟标记名，或是离散或模拟表达式。

附注

触发器标记仅当 InfoResources() 函数用在动画显示链接中时才有意义。如果此函数用在脚本中，则可以指定任何数值、模拟型标记名或数值表达式。

示例

此脚本检索 Windows 任务的大致数量，并且如果用在动画显示链接中，还每秒钟更新一次信息。

```
InfoResources(2,$second);
```

检索 InTouch 相关信息

在脚本中，可以使用以下函数来检索 InTouch 相关信息。

| 函数 | 作用 |
|---------------------|--------------------------------|
| InfoInTouchAppDir() | 获取正在开发的 InTouch 应用程序所在目录的有关信息。 |
| InTouchVersion() | 获取 InTouch 版本信息。 |

检索 InTouch 应用程序目录的名称

在脚本中，通过使用 InfoInTouchAppDir() 函数，可以检索 InTouch 应用程序所在目录的名称。对于查找要随您的 InTouch 应用程序一起交付的任何外部文件，此函数很有用。

InfoInTouchAppDir() 函数

返回当前 InTouch 应用程序目录。

语法

```
result = InfoInTouchAppDir();
```

返回值

包含当前运行的 InTouch 应用程序所在目录的消息标记名。

附注

由于 131 个字符的限制，传递给消息标记名或在动画链接中显示时，应用程序目录名可能被截断。

示例

此脚本可能返回 c:\documents and settings\user1\my documents\my intouch applications\packaging。

```
InfoInTouchAppDir()
```

检索 InTouch 版本

在脚本中，通过使用 `InTouchVersion()` 函数可以检索当前正在运行的 InTouch 应用程序的版本号。

InTouchVersion() 函数

返回完整的 InTouch 版本号或其一部分。

语法

`result = InTouchVersion (infotype);`

参数

infotype

指定如何返回版本信息。含义如下的整数值、整型标记名或整型表达式：

- 0- 函数返回整个版本号
- 1- 函数仅返回主版本号
- 2- 函数仅返回次版本号
- 3- 函数仅返回补丁级别
- 4- 函数仅返回内部版本级别

示例

| 函数 | 可能的结果 |
|--------------------------------|---------------------------|
| <code>InTouchVersion(0)</code> | 9.5.0 1101.0377.0093.0031 |
| <code>InTouchVersion(1)</code> | 9 |
| <code>InTouchVersion(2)</code> | 5 |
| <code>InTouchVersion(3)</code> | 0 |
| <code>InTouchVersion(4)</code> | 1101 |

安全性相关脚本

您可以使用各种 QuickScript 函数与系统标记在 InTouch 中添加与管理安全性。如需有关安全性函数的详细信息，请参阅 *InTouch® HMI 应用程序管理与扩展指南* 中的第 5 章 “保护 InTouch 安全”。

登录与注销

您可以使用以下函数与系统标记进行登录与注销。

| 函数 | 作用 |
|-------------------------|--|
| AttemptInvisibleLogon() | 通过在参数中提供身份验证数据将用户登录进来。 |
| LogonCurrentUser() | 将当前已登录到 Windows 的用户（如果身份验证模式为 "OS"）登录进来。 |
| PostLogonDialog() | 显示 登录 对话框。 |
| Logoff() | 注销当前用户。 |
| \$PasswordEntered | 设置口令。 |
| \$OperatorEntered | 设置有效的用户名。 |
| \$OperatorDomainEntered | 设置有效的用户域名（如果身份验证模式为 "OS"）。 |

如需有关安全性函数的详细信息，请参阅 *InTouch® HMI 应用程序管理与扩展指南* 中的第 5 章 “保护 InTouch 安全”。

更改与设置口令

您可以使用以下函数与系统标记来更改口令：

| 函数 | 作用 |
|------------------|-----------------------------|
| ChangePassword() | 为当前登录的用户调用 改变口令 对话框。 |
| \$ChangePassword | 为当前登录的用户调用 改变口令 对话框。 |

如需有关安全性函数的详细信息，请参阅 *InTouch® HMI 应用程序管理与扩展指南* 中的第 5 章 “保护 InTouch 安全”。

指定与配置用户

您可以使用以下系统标记指定与配置用户。

| 函数 | 作用 |
|------------------|------------|
| \$ConfigureUsers | 调用配置用户对话框。 |

如需有关安全性函数的详细信息，请参阅 *InTouch® HMI 应用程序管理与扩展指南* 中的第 5 章 “保护 InTouch 安全”。

管理安全性及其它信息

您可以使用以下系统标记与函数来管理安全性。

| 函数 | 作用 |
|------------------------------|---------------------------|
| \$AccessLevel | 检索当前登录的用户的访问级别。 |
| AddPermission() | 将访问级别指定给特定的用户组（本地 / 域）。 |
| GetAccountStatus() | 检索帐户信息（口令过期、锁定、禁用标识）。 |
| \$InactivityTimeout | 显示在用户自动注销之前经过的时间。 |
| \$InactivityWarning | 显示超时警告的时间。 |
| InvisibleVerifyCredentials() | 检索操作系统用户的 InTouch 访问级别信息。 |
| IsAssignedRole() | 查看当前登录的用户是不是特定用户角色的成员。 |
| QueryGroupMembership() | 查看当前登录的用户是不是特定用户角色的成员。 |

如需有关安全性函数的详细信息，请参阅 *InTouch® HMI 应用程序管理与扩展指南* 中的第 5 章 “保护 InTouch 安全”。

其它脚本

InTouch 脚本支持声音输出，这样就可以通过声音进行人机交互。InTouch 脚本也支持获取与设置“向导”属性。

从应用程序 InTouch 播放声音文件

在脚本中，可以将事件与条件同特定的声音关联起来。例如，可以将警告对话框或临界条件与警告声音关联起来。

PlaySound() 函数

播放波形文件的声音或 Windows 缺省声音。

语法

Playsound (*soundname*, *flag*)

参数

soundname

声音或波形文件的名称。字符串值、消息标记名或字符串表达式。如果给声音定义名称，它必须在 Win.ini 文件中的 [Sounds] 部分进行定义，例如 MC=" c:\test.wav"

flag

指定如何播放声音。文字整数值、整型标记名或整型表达式，含义如下：

- 0 - 同步播放声音一次（脚本等到声音播放完毕才继续执行）。
- 1 - 异步播放声音一次（脚本不必等待声音播放完毕再继续执行）。
- 9 - 连续播放声音（直到再次调用 PlaySound() 函数为止）。

示例

此脚本播放 c:\welcome.wav 文件的声音一次，并暂停脚本执行，直到声音播放完毕。

```
PlaySound("c:\welcome.wav",0);
```

此脚本连续播放声音警告。在 win.ini 文件 [Sounds] 部分，需要将声音名称 Alert 与声音文件关联起来，例如：

```
Alert=c:\alert.wav.
```

```
PlaySound("Alert",9);
```

获取与设置向导属性

有些向导（如“分布式报警对象”与 Windows 控件）包含可设置或读取的属性。这些属性可以是文本框中的值或是复选框的选取状态。

在脚本中，通过以下函数可以访问这些属性。

| 函数 | 作用 |
|-----------------------------------|------------|
| SetPropertyD(), GetPropertyD() | 设置或读取离散属性。 |
| SetPropertyI(), GetPropertyI() | 设置或读取整型属性。 |
| SetPropertyM(), GetPropertyM() | 设置或读取消息属性。 |

如需所支持的属性的列表，请参阅向导描述。

下面是如何设置与读取这些属性的一般方法。

GetPropertyD() 函数

读取向导中的离散属性并返回成功码。

语法

result = GetPropertyD (controlname.property, dtag)

参数

controlname

支持属性的向导的名称。字符串值、消息标记名或字符串表达式。

property

向导中要读取的离散属性。与 *controlname* 合起来，可以是字符串值、消息标记名或字符串表达式。

dtag

将接收离散属性值的离散标记名。

返回值

整型错误码。如需有关错误码的详细信息，请参阅 *InTouch® HMI 可视化指南* 中的第 5 章“向导”中的“理解 Windows 控件错误消息”。

示例

通过复选框向导 *Checkbox1* 与离散标记名 *dtag*，可以使用以下脚本函数来检查复选框的可见性：

```
result=GetPropertyD("Checkbox1.visible",dtag);
```

如果复选框为可见，此脚本将 *dtag* 设为 1；否则它将 *dtag* 设为 0。

SetPropertyD() 函数

设置向导中的离散属性并返回成功码。

语法

```
result = SetPropertyD(controlname.property, Boolean)
```

参数

controlname

支持属性的向导的名称。字符串值、消息标记名或字符串表达式。

property

向导中要设置的离散属性。与 *controlname* 合起来，可以是字符串值、消息标记名或字符串表达式。

Boolean

要传递给向导属性的布尔值。布尔值、离散标记名或布尔表达式。

返回值

整型错误码。如需有关错误码的详细信息，请参阅 *InTouch*® *HMI 可视化指南* 中的第 5 章“向导”中的“理解 Windows 控件错误消息”。

示例

通过复选框向导 *Checkbox1* 与离散标记名 *dtag*，可以使用以下脚本函数来控制复选框的可见性：

```
result=SetPropertyD("Checkbox1.visible",dtag);
```

如果将 *dtag* 设为 0 并调用上面的脚本函数，复选框向导会消失。

GetPropertyI() 函数

读取向导中的某个整数并返回成功码。

语法

```
result = GetPropertyI (controlname.property, itag)
```

参数

controlname

支持属性的向导的名称。字符串值、消息标记名或字符串表达式。

property

向导中要读取的整型属性。与 *controlname* 合起来，可以是字符串值、消息标记名或字符串表达式。

itag

将接收整型属性值的整型标记名。

返回值

整型错误码。如需有关错误码的详细信息，请参阅 *InTouch® HMI 可视化指南* 中的第 5 章“向导”中的“理解 Windows 控件错误消息”。

示例

通过单选按钮向导 *Radiobutton1* 与整型标记名 *itag*，可以使用以下脚本函数来检查单选按钮组中当前所选的项目：

```
result=GetPropertyI("Radiobutton1.value",itag);
```

如果选择了第一个（第二个、第三个、...）单选按钮，则此脚本将 *itag* 设为 1 (2, 3, ...).

SetPropertyI() 函数

设置向导中的整型属性并返回成功码。

语法

```
result = SetPropertyI (controlname.property, integer)
```

参数

controlname

支持属性的向导的名称。字符串值、消息标记名或字符串表达式。

property

向导中要设置的整型属性。与 *controlname* 合起来，可以是字符串值、消息标记名或字符串表达式。

integer

要传递给向导属性的整数值。整数值、整型标记名或整型表达式。

返回值

整型错误码。如需有关错误码的详细信息，请参阅 *InTouch® HMI 可视化指南* 中的第 5 章“向导”中的“理解 Windows 控件错误消息”。

示例

通过单选按钮向导 *Radiobutton1*，可以使用以下脚本函数来设置第二个单选按钮：

```
result=SetPropertyI("Radiobutton1.value",2);
```

GetPropertyM() 函数

读取向导中的消息属性并返回成功码。

语法

```
result = GetPropertyM (controlname.property, mtag)
```

参数

controlname

支持属性的向导的名称。字符串值、消息标记名或字符串表达式。

property

向导中要读取的消息属性。与 *controlname* 合起来，可以是字符串值、消息标记名或字符串表达式。

mtag

将接收消息属性值的消息标记名。

返回值

整型错误码。如需有关错误码的详细信息，请参阅 *InTouch® HMI 可视化指南* 中的第 5 章“向导”中的“理解 Windows 控件错误消息”。

示例

通过复选框向导 *Checkbox1* 与消息标记名 *mtag*，可以使用以下脚本函数来检查复选框的标题：

```
result=GetPropertyM("Checkbox1.caption",mtag);
```

此脚本将 *mtag* 设为复选框的标题。

SetPropertyM() 函数

设置向导中的消息属性并返回成功码。

语法

```
result = SetPropertyM (controlname.property, message)
```

参数

controlname

支持属性的向导的名称。字符串值、消息标记名或字符串表达式。

property

向导中要设置的消息属性。与 *controlname* 合起来，可以是字符串值、消息标记名或字符串表达式。

message

要传递给向导属性的消息值。字符串值、消息标记名或字符串表达式。

返回值

整型错误码。如需有关错误码的详细信息，请参阅 *InTouch® HMI 可视化指南* 中的第 5 章“向导”中的“理解 Windows 控件错误消息”。

示例

通过复选框向导 *Checkbox1*，可以使用以下脚本函数来动态设置复选框向导的标题：

```
result=SetPropertyM("Checkbox1.caption","Start Engine 1");
```

此脚本将 *Checkbox1* 复选框的标题设为 “Start Engine 1”。

第 7 章

使用 OLE 对象编写脚本

您可以使用 OLE 对象来扩展 InTouch HMI 应用程序的功能。使用 OLE 对象可以：

- 为操作员界面创建弹出式对话框。
- 访问操作系统功能，如“控制面板”。
- 使“制造执行模块”中的数据可以在 InTouch HMI 中处理。请参阅“制造执行模块”文档。

创建、验证及释放 OLE 对象

您可以创建与验证 OLE 对象以便在 InTouch 脚本中使用。使用 OLE 对象之后，可以释放它以释放内存。

使用以下函数来创建、验证及释放 OLE 对象。

- OLE_CreateObject() 函数
- OLE_IsObjectValid() 函数
- OLE_ReleaseObject() 函数

OLE_CreateObject() 函数

必须先创建 OLE 对象，然后才能在脚本中引用它。创建时会得到引用该 OLE 对象的指针。

在脚本中，可以通过使用 OLE_CreateObject() 函数来创建 OLE 对象并指定指针。

语法

```
OLE_CreateObject(%pointer, classname);
```

参数

%pointer

为 OLE 对象所选择的指针的名称。它可以包含字母数字字符（A-Z、0-9）与下划线，它不区分大小写。

classname

OLE 类的名称。类名区分大小写。字符串值、消息标记名或字符串表达式。

附注

如果使用相同的对象名创建另一个对象，则该对象更新为引用新的 OLE 类。它会从旧的 OLE 类中释放掉。

示例

此脚本创建一个 OLE 对象 %WShell，该对象引用 Wscript.Shell 类。

```
OLE_CreateObject(%WShell, "Wscript.Shell");
```

OLE_IsObjectValid() 函数

在脚本中，可以使用 OLE_IsObjectValid() 函数来验证 OLE 对象是否有效。对于使用 OLE 对象，这并不是一个必需的步骤，但为了确保使用 OLE 对象时不会遇到问题，建议执行该步骤。

语法

```
result = OLE_IsObjectValid(%pointer)
```

参数

%pointer

引用要测试的 OLE 对象的指针。

result

“布尔”值，具体如下：

0 - 指针引用的 OLE 对象无效。

1 - 指针引用的 OLE 对象有效。

示例

此脚本基于 *Wscript.Shell* 类创建一个 OLE 对象，并创建引用该对象的指针 %WS。 *isvalid* 是离散标记，如果成功创建 OLE 对象则为 TRUE。否则为 FALSE。

```
OLE_CreateObject(%WS, "Wscript.Shell");  
isvalid = OLE_IsObjectValid(%WS);
```

OLE_ReleaseObject() 函数

在脚本中使用 OLE 对象之后，可以释放它并删除其指针，以释放系统资源。释放 OLE 对象之后，不能使用其指针来访问关联的 OLE 类的属性和方法。

语法

```
OLE_ReleaseObject(%pointer);
```

参数

%pointer

引用 OLE 对象的指针的名称。它可以包含字母数字字符（A-Z、0-9）与下划线，它不区分大小写。

示例

此脚本释放与指针 %WShell 关联的 OLE 对象，并删除指针 %WShell。

```
OLE_ReleaseObject(%WShell);
```

使用 OLE 对象的属性与方法

在脚本中，可以使用指针对 OLE 属性中的值进行读写。您也可以使用指针调用 OLE 方法。属性与方法是否可用取决于 OLE 对象。

访问 OLE 对象的属性

在脚本中，可以像访问大多数编程语言那样访问 OLE 对象的属性。属性通常使用点 “.” 运算符来确定。

备注 在脚本中使用 OLE 对象属性时，确保其引用不超过 98 个字符（包括前导字符 “%”）。尽可能保持 OLE 指针名简短一些。

读取 OLE 对象属性

在脚本中，可以通过将属性赋值给标记来读取 OLE 对象属性。在动画显示链接中，无法将直接引用用于 OLE 对象属性。

语法

```
value = %pointer.property;
```

参数

%pointer

引用 OLE 对象的指针。在读取属性之前，必须使用 OLE_CreateObject() 函数创建，或赋给另一个指针。

property

要读取的属性的名称。

value

要写入值的标记。

示例

此脚本基于 *System.Random* OLE 类创建 OLE 对象，创建引用它的指针 *%SR*，并将 Math.Random OLE 对象的 *.NextDouble* 属性的值指定给实型标记名 *randtag*。

在运行时，实型标记名 *Randtag* 接收到 0 到 1 之间的随机双精度浮点值。

```
OLE_CreateObject(%SR,"System.Random");  
randtag = %SR.NextDouble;
```

写入 OLE 对象属性

在脚本中，可以通过将值赋给属性来将它写入 OLE 对象属性。

语法

```
%pointer.property = value;
```

参数

%pointer

引用 OLE 对象的指针。在写入属性之前，必须使用 OLE_CreateObject() 函数创建，或赋给另一个指针。

property

要写入的属性的名称。

value

要写入属性的值。它可以数值、标记名或表达式。不支持直接从动画输入链接写入 OLE 属性。

调用 OLE 对象的方法

在脚本中，可以调用 OLE 对象方法。

语法

```
%pointer.method(parameters);
```

参数

%pointer

引用 OLE 对象的指针。在调用方法之前，必须使用 OLE_CreateObject() 函数创建，或赋给另一个指针。

method

属于 OLE 对象一部分的方法的名称。

parameters

传递给该方法的参数的列表。这些参数必须使用逗号隔开。值、标记名或表达式。

示例

此脚本基于 OLE 类 *Shell.Application* 创建一个 OLE 对象，创建引用该 OLE 对象的指针 *%sa*，并调用方法 *.MinimizeAll()*。此方法最小化桌面上的所有窗口。

```
OLE_CreateObject(%SA,"Shell.Application");  
%SA.MinimizeAll();
```

备注 OLE InTouch HMI 脚本中不允许使用可选参数。所有参数都必须指定。

将多个指针指定给相同的 OLE 对象

在脚本中通过使用等号，可以将多个指针指定给相同的 OLE 对象。

语法

```
%newpointer = %pointer
```

参数

%pointer

已引用某个已创建的 OLE 对象的指针的名称。

%newpointer

应该引用相同 OLE 对象的新指针的名称。它可以包含字母数字字符（A-Z、0-9）与下划线，它不区分大小写。

示例

此脚本基于 *Wscript.Shell* 类创建一个 OLE 对象，并创建引用该对象的指针 *%WS*。指针 *%WS2* 设置为 *%WS* 时指向相同的 OLE 对象。它可以用于读取或写入相同 OLE 对象的属性及调用其方法。

```
OLE_CreateObject(%WS,"Wscript.Shell");  
%WS2=%WS;
```

备注 您可以使用与指针相关的消息标记名。如果将消息标记名赋给指针，则它可以获取一个 ID 值。您可以使用它来创建更多指向相同 OLE 对象的指针。

排解 OLE 错误

在脚本中，可以使用 OLE 函数来排解 OLE 错误。

| 函数 | 描述 |
|------------------------------------|---------------------------------|
| OLE_GetLastObjectError() 函数 | 获取上一个 OLE 错误的错误号。 |
| OLE_GetLastObjectErrorMessage() 函数 | 获取有关上一个 OLE 错误的信息。 |
| OLE_ResetObjectError() 函数 | 复位上一个错误。 |
| OLE_ShowMessageOnObjectError() 函数 | 显示或隐藏 OLE 错误消息对话框。 |
| OLE_IncrementOnObjectError() 函数 | 统计含 InTouch HMI 标记名的 OLE 错误的数量。 |

OLE_GetLastObjectError() 函数

此函数返回上一个 OLE 错误的错误号。

语法

```
errnum = OLE_GetLastObjectError();
```

参数

errnum

上一个 OLE 错误的编号。

OLE_GetLastObjectErrorMessage() 函数

此函数返回上一个 OLE 错误的错误消息。

语法

```
errmsg = OLE_GetLastObjectErrorMessage();
```

参数

errmsg

上一个 OLE 错误的错误消息。

OLE_ResetObjectError() 函数

在脚本中，使用 OLE_ResetObjectError() 函数复位上一个 OLE 错误，使上一个 OLE 错误号设置为零，且上一个 OLE 错误消息设置为空白。

这可以用于确定一批 OLE 函数中的任何错误。

语法

```
OLE_ResetObjectError()
```

OLE_ShowMessageOnObjectError() 函数

缺省条件下，发生 OLE 错误时，显示一个错误消息对话框。

在脚本中，通过使用函数

`OLE_ShowMessageOnObjectError()`，可以指定是否显示错误消息对话框。

语法

`OLE_ShowMessageOnObjectError(Boolean)`

参数

Boolean

用于确定是否显示 OLE 错误消息对话框的值。布尔值、离散标记名或布尔表达式，含义如下：

- 0 - 发生 OLE 错误时不显示 OLE 错误消息对话框
- 1 - 发生 OLE 错误时显示 OLE 错误消息对话框

示例

此脚本抑制所有的 OLE 错误消息对话框。发生 OLE 错误时，不显示任何错误消息对话框。

```
OLE_ShowMessageOnObjectError(0);
```

OLE_IncrementOnObjectError() 函数

在脚本中，可以使用 `OLE_IncrementOnObjectError()` 函数将一个整型标记名指定为 OLE 错误数量的计数器。

语法

`OLE_IncrementOnObjectError(integertag)`

参数

integertag

充当计数器的标记名。

附注

如果显示 OLE 错误消息对话框，则计数器标记名仅在 OLE 错误消息对话框关闭之后递增。

示例

此脚本将整型标记名 *errorcount* 指定为错误计数器，隐藏错误消息对话框，并尝试基于无效的 OLE 类名来创建一个 OLE 对象。这会导致错误，且标记名值 *errorcount* 递增到 1。

```
errorcount = 0;  
OLE_IncrementOnObjectError(errorcount);  
OLE_ShowMessageOnObjectError(0);  
OLE_CreateObject(%WS,"InVaLiD.cLaSs.nAmE");
```


使用 OLE 的好处

使用 OLE 对象可以给应用程序添加强大的功能；通过以下脚本，您将对此有所认识。

随机号产生随机数

在脚本中，使用以下命令产生 0 到 255 之间的一个随机数：

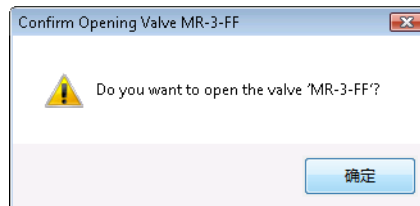
```
OLE_CreateObject(%SR,"System.Random");  
randtag = (%SR.NextDouble)*255;
```

创建用户界面对话框





在脚本中，使用以下命令产生用户界面对话框：

```
dim DlgBody as message;  
dim DlgTitle as message;  
dim Style as integer;  
dim Result as integer;  
  
DlgBody = "Do you want to open the valve ' MR-3-FF' ?";  
DlgTitle = "Confirm Opening Valve MR-3-FF";  
Style = 48;  
  
OLE_CreateObject(%WS,"Wscript.Shell");  
result = %WS.Popup(DlgBody,1,DlgTitle,Style);
```

本例创建以下用户界面对话框。



Style 标记名确定有哪些图标与按钮出现在对话框上。使用以下值：

| 图标 | 样式 | 值 |
|---|------|----|
| (无图标) | 无图标 | 0 |
|  | 错误图标 | 16 |
|  | 问号图标 | 32 |
|  | 警告图标 | 48 |
|  | 信息图标 | 64 |

要使用特定的按钮，请将以下值之一添加到 Style 值：

| 值 | 样式 |
|---|------------|
| 0 | 仅确定按钮 |
| 1 | 确定与取消按钮 |
| 2 | 放弃、重试及忽略按钮 |
| 3 | 是、否及取消按钮 |
| 4 | 是与否按钮 |
| 5 | 重试与取消按钮 |
| 6 | 取消、重试及继续按钮 |

Result 标记名包含用户单击的按钮编号。这可用作 InTouch 脚本中的条件分支。可能的结果码如下：

| 结果值 | 含义 |
|-----|-----------------|
| 1 | 按了 确定 按钮 |
| 2 | 按了 取消 按钮 |
| 3 | 按了 放弃 按钮 |
| 4 | 按了 重试 按钮 |
| 5 | 按了 忽略 按钮 |
| 6 | 按了 是 按钮 |
| 7 | 按了 否 按钮 |
| 10 | 按了 重试 按钮 |
| 11 | 按了 继续 按钮 |

打开 Windows 日期与时间属性面板

在脚本中，使用以下命令打开 Windows 的“日期/时间属性”面板：

```
OLE_CreateObject(%WP,"Shell.Application");
%WP.SetTime();
```

通过调用不同的方法并将它们传递给所引用的 OLE 对象，可以执行类似的任务：

| 这个方法 | 打开面板 |
|-------------------|---------------|
| TrayProperties() | 工具栏属性 |
| FileRun() | 文件运行对话框 |
| FindFiles() | 查找文件对话框 |
| FindComputer() | 查找计算机对话框 |
| ShutdownWindows() | 关闭 Windows 面板 |

读取和写入注册表

在脚本中，可以通过以下方法使用 OLE 对 Windows 注册表进行读写：

- 基于 Windows 类 Wscript.Shell 创建 OLE 对象。
- 使用 OLE 对象的 RegRead() 与 RegWrite() 方法。

例如，这些命令直接从注册表键读取安装的 InTouch HMI 的版本，然后将值存储到 rkey 消息标记名：

```
OLE_CreateObject(%WS,"Wscript.Shell");
rkey =
    %WS.RegRead("HKLM\SOFTWARE\Wonderware\InTouch\Installation\Version");
```

这些命令将值 1 写入某个注册表键，该键确定对于当前登录的用户，文件扩展名是否隐藏：

```
OLE_CreateObject(%WS,"Wscript.Shell");
%WS.RegWrite("HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\HideFileExt",1,"REG_DWORD");
```

最小化窗口

在脚本中，可以使用以下命令最小化桌面上的所有窗口：

```
OLE_CreateObject(%WA,"Shell.Application");
%WA.MinimizeAll();
```

通过调用这些方法，可以执行一些类似的任务：

| 这个方法 | 排列窗口 |
|--------------------|----------|
| TileHorizontally() | 横向平铺所有窗口 |
| TileVertically() | 纵向平铺所有窗口 |
| CascadeWindows() | 层叠所有窗口 |
| UndoMinimizeALL() | 还原所有窗口 |

第 8 章

编写 ActiveX 控件脚本

您可以使用 ActiveX 控件读取和写入标记名与 I/O 引用。在脚本中，可以引用 ActiveX 控件。

您也可以创建在 ActiveX 控件发生事件时执行的脚本。这些脚本可以复用，也可以导入到其它应用程序中。

调用 ActiveX 控件方法

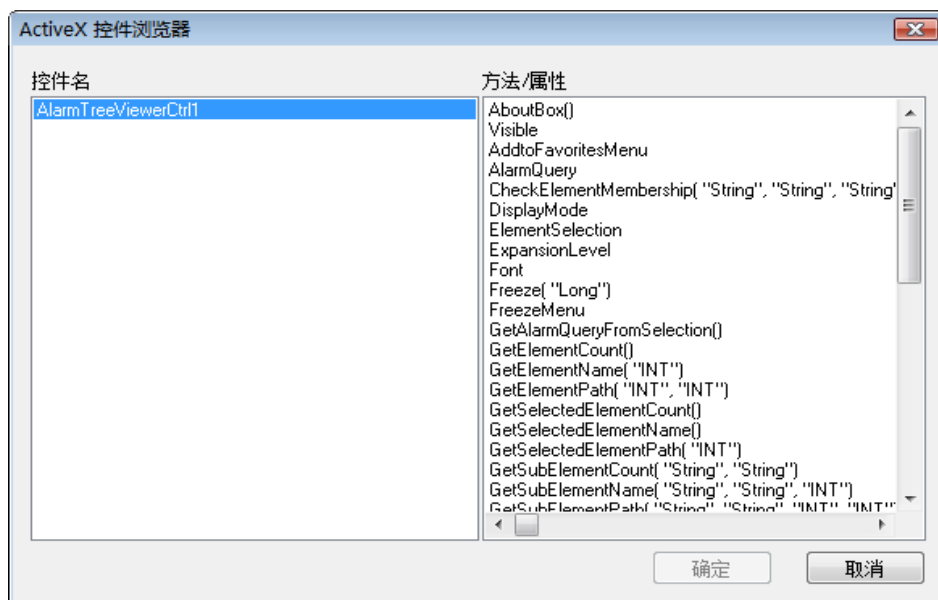
在脚本中，可以调用 ActiveX 控件的方法以执行 ActiveX 控件所支持的操作。ActiveX 方法可以从任何类型的 InTouch QuickScript 或 ActiveX 事件脚本中调用。

备注 要在发生 ActiveX 事件时调用 ActiveX 方法，需要执行一些必要的操作。请参阅第 34 页的“配置 ActiveX 事件脚本”。

要调用 ActiveX 控件方法

- 1 在脚本对话框中的**插入**菜单上，单击 **ActiveX**。

此时出现 **ActiveX 控件浏览器**对话框。



- 2 从左侧窗格中单击 **ActiveX** 控件的名称。此时右侧窗格包含该 **ActiveX** 控件所支持的属性与方法的名称。
- 3 从右侧窗格中单击要使用的方法的名称，然后单击**确定**。此时方法名与缺省参数粘贴到脚本窗口中的光标位置。
- 4 根据规格配置括号内的方法参数。

从 InTouch HMI 访问 ActiveX 控件属性

在脚本中，可以读取和写入 ActiveX 控件属性，以便在 ActiveX 控件与 InTouch 标记名及显示链接之间交换数据。

配置 ActiveX 控件属性以读取和写入数据

在脚本中，可以对 ActiveX 控件进行数据的读取和写入。您使用与特定的 ActiveX 控件关联的 ActiveX 控件属性。

实现方法有两种：

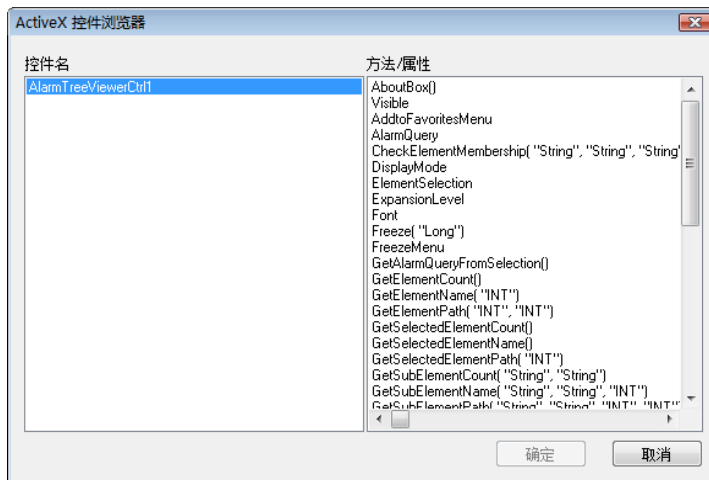
- 在 InTouch HMI QuickScript 或 ActiveX 事件脚本中使用 ActiveX 控件属性。属性值在每次执行脚本时读取或写入。
- 将 ActiveX 控件属性直接链接到 InTouch HMI 标记或 I/O 引用。属性值在每个更新间隔进行读取或写入。

配置脚本以读取和写入 ActiveX 控件属性

在脚本中，可以配置 ActiveX 控件属性以便将值写入 InTouch HMI 标记名或其它表达式，或是从中读取值。

要对 ActiveX 控件属性进行数据的读取或写入

- 1 打开脚本窗口，指向**插入**，然后单击 **ActiveX**。此时出现 **ActiveX 控件浏览器**对话框。



- 2 从左侧窗格中单击 ActiveX 控件的名称。此时右侧窗格包含所选 ActiveX 控件的属性与方法名称。
- 3 从右侧窗格中单击要使用的属性的名称。此时属性名插入到脚本窗口中的光标位置。
- 4 将属性名指定给标记，或根据您的规格来使用。
- 5 单击**确定**。

示例

以下脚本将 ActiveX 控件实例 *AlarmViewerCtrl1* 的 *ToPriority* 属性读取到整型标记名 *topri* 中。

```
topri = #AlarmViewerCtrl1.ToPriority;
```

以下脚本将 MS Comic 这个值写入 AlarmViewerCtrl1 这个 ActiveX 控件的 Font 属性。本例动态更改 AlarmViewer ActiveX 控件的显示字体。

```
#AlarmViewerCtrl1.Font = "MS Comic";
```

将 ActiveX 控件属性链接到标记或 I/O 引用

您可以将 ActiveX 控件属性链接到 InTouch HMI 标记或 I/O 引用。

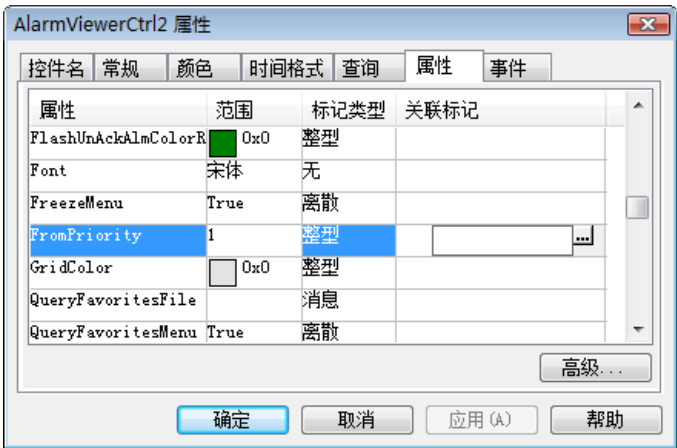
要将 ActiveX 控件属性链接到标记或 I/O 引用。

- 1 双击 ActiveX 控件。此时出现该 ActiveX 控件的属性对话框。



- 2 单击**属性**选项卡并滚动到右侧。

3 选择列表中的属性。



- 4 指定标记或 I/O 引用。执行以下操作之一：
- 将标记或 I/O 引用直接输入到**关联标记**列。
 - 单击方括号之间的**关联标记**列中的省略号按钮。此时出现**选择标记**对话框。选择标记，然后单击**确定**。
- 5 单击**确定**。

创建与复用 ActiveX 事件脚本

ActiveX 控件可以支持事件，例如可用于关联特定操作的控件单击事件。这些操作存储在 ActiveX 事件脚本中。

创建 ActiveX 事件脚本

您可以创建或复用在每次发生特定的 ActiveX 控件事件（如单击 ActiveX 控件）时执行的事件脚本。

要创建 ActiveX 事件脚本

- 1 双击 ActiveX 控件。此时出现属性对话框。
- 2 单击事件选项卡。



- 3 单击要关联的事件。此时括号与省略号出现在脚本列中。



- 4 在相应行的脚本列中，单击括号之间。
- 5 输入新的名称，然后单击确定。出现消息时，单击确定。此时出现 ActiveX 事件脚本对话框。
- 6 根据您的规格创建脚本。

复用 ActiveX 事件脚本

如果 ActiveX 事件脚本由相同的父 ActiveX 控件与事件创建，便可以复用它们。

例如，如果应用程序中有多个 AlarmViewer ActiveX 控件，则它们可以共享 DoubleClick 事件的事件脚本。

要复用 ActiveX 事件脚本

- 1 双击 ActiveX 控件。此时出现属性对话框。
- 2 单击**事件**选项卡。
- 3 单击要关联的事件。此时括号与省略号出现在**脚本**列中。



- 4 在相应行的**脚本**列中，单击省略号按钮。此时出现**选择 ActiveX 脚本**对话框。
- 5 单击 ActiveX 脚本，然后单击**确定**。
- 6 再次单击**确定**。

创建自引用 ActiveX 事件脚本

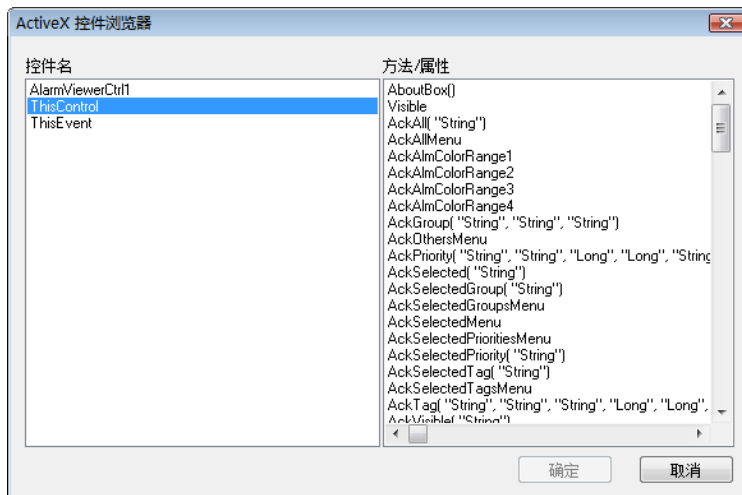
如果使用 ActiveX 事件脚本，则可以将它们配置为引用自身，而不是引用绝对 ActiveX 控件名。这在创建将要复用的 ActiveX 事件脚本时很有用处。ActiveX 事件脚本可以：

- 引用产生事件的特定 ActiveX 控件 (ThisControl)。
- 引用调用脚本的特定事件 (ThisEvent)。

通过引用特定事件，可以让 ActiveX 控件将其它参数传递给 ActiveX 控件脚本。

要创建自引用的 ActiveX 事件脚本

- 1 为特定的 ActiveX 事件创建 ActiveX 事件脚本。请参阅第 169 页的“创建 ActiveX 事件脚本”。
- 2 在 **ActiveX 事件脚本** 对话框中，单击**插入**，然后单击**ActiveX**。此时出现 **ActiveX 控件浏览器** 对话框。



- 3 在左侧窗格中，执行以下操作之一：
 - 单击 **ThisControl**，以查看可用于连接此控件（以及在其中复用此脚本的任何其它控件）的属性与方法。
 - 单击 **ThisEvent**，以查看可用于连接自引用事件的 ActiveX 控件的属性与方法。
- 4 在右侧窗格中，单击属性或方法之一，然后单击**确定**。此时所选的属性或方法粘贴到脚本窗口中。
- 5 配置脚本。
- 6 单击**确定**。

例如，此语句将 ClicknRow 事件参数的值写入 ClickedRow 标记：

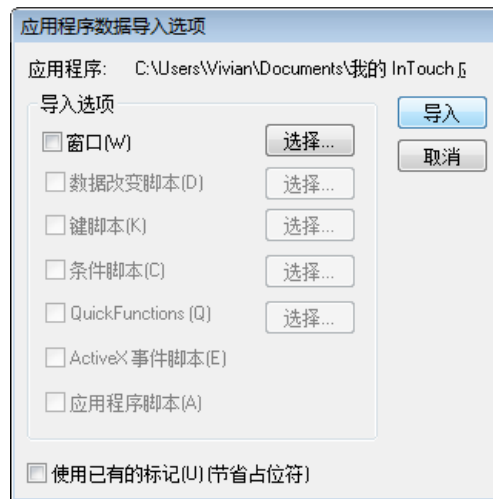
```
ClickedRow = ThisEvent.ClicknRow;
```

导入 ActiveX 事件脚本

您可以从其它 InTouch HMI 应用程序中导入 ActiveX 事件脚本，以便在当前正在开发的应用程序中复用它们。

要从其它应用程序导入 ActiveX 事件脚本

- 1 在文件菜单上，单击导入。此时出现自目录导入对话框。
- 2 浏览到包含要导入的 ActiveX 事件脚本的 InTouch HMI 应用程序。
- 3 单击确定。此时出现应用程序数据导入选项对话框。



- 4 选择 **ActiveX 事件脚本** 复选框，然后单击导入。此时所有的 ActiveX 事件脚本都导入到当前的 InTouch HMI 应用程序中。

第 9 章

QuickScript 疑难排解

通过使用 Log Viewer 来显示标记名的运行时值，可以排解 QuickScript 的问题。

将消息记录到 Log Viewer

使用 ArchestrA Log Viewer 帮助调试 QuickScript。Log Viewer 在 ArchestrA 系统管理控制台（ArchestrA 系统管理控制台，简称 SMC）中，它随同 InTouch HMI 一起安装。

调试 QuickScript 的一种方法是：

- 1 在 QuickScript 中设置检查点，以便将一些值记录到 Log Viewer。
- 2 打开 Log Viewer 来查看这些值。

另一种方法是创建一个“键脚本”，将标记值记录到 Log Viewer。

要在 QuickScript 中设置检查点

- 1 打开怀疑可能导致错误的 QuickScript。
- 2 找到希望设置检查点的行。
- 3 将下面的代码段之一插入该行的后面：
 - `LogMessage(messagetag);`
在这个脚本中，*messagetag* 是您希望记录其值的消息标记名的名称。
 - `LogMessage(StringFromIntg(inttag,10));`
在这个脚本中，*inttag* 是您希望记录其值的整型标记名的名称。
 - `LogMessage(Text(realtag,"#.#####"));`
在这个脚本中，*realtag* 是您希望记录其值的实型标记名的名称。
 - `LogMessage(DText(disctag," TRUE"," FALSE"));`
在这个脚本中，*disctag* 是您希望记录其值的离散标记名的名称。
 - 在一个检查点上记录更多的信息到 LogViewer，如标识符与 / 或标记名。例如，

```
LogMessage("DEBUG tag:"+ind.name+"
value:"+Text(ind,"#.####"));
```

在这个脚本中，*ind* 可以是间接模拟标记。

LogMessage() 函数

将用户自定义的消息写入 ArchedrA Log Viewer。

类别

其它

语法

```
LogMessage("Message_Tag");
```

参数

Message_Tag

要记录到 Log Viewer 的字符串 实际的字符串或消息标记名。

附注

对于排解 InTouch 脚本的问题来说，这是一个功能非常强大的函数。通过策略性地将 LogMessage() 函数放入脚本，可以判断 QuickScript 的执行顺序、脚本的性能，以及确定在标记发生更改之前和受 QuickScript 影响之后的值。发送到 Log Viewer 的每条消息都会加上准确的日期与时间标签。

示例

```
LogMessage("Report Script is Running");
```

上面的语句会将以下内容打印到 Log Viewer:

```
94/01/14 15:21:14 WWSCRIPT Message:Report Script is Running.
```

```
LogMessage("The Value of MyTag is " + Text(MyTag, "#"));
```

```
MyTag = MyTag + 10;
```

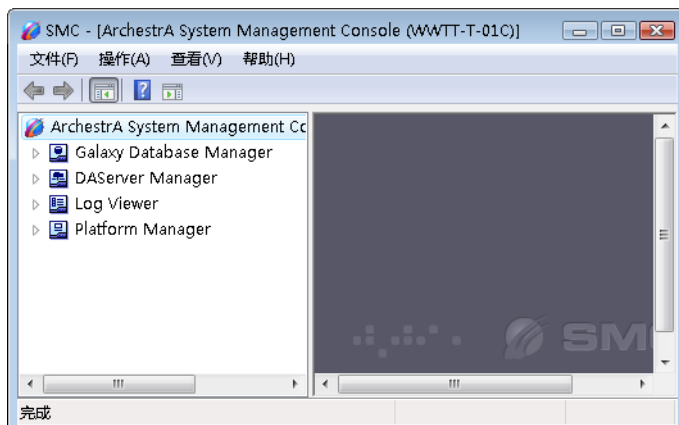
```
LogMessage("The Value of MyTag is " + Text(MyTag, "#"));
```

查看 Log Viewer 消息

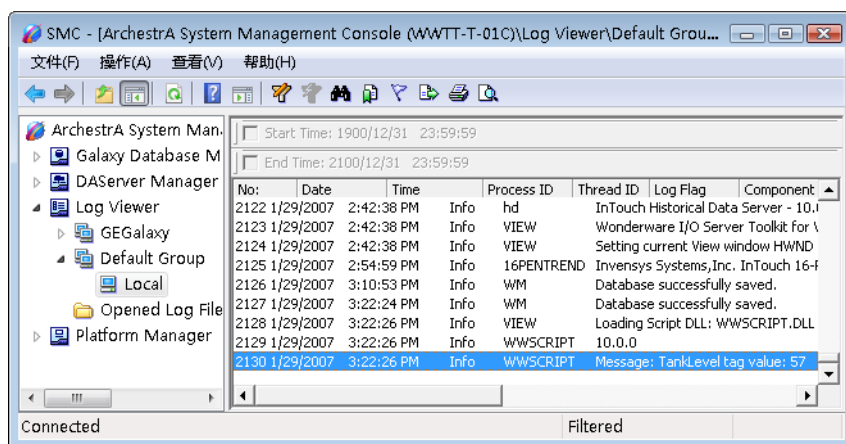
Log Viewer 在 ArchestrA 系统管理控制台（ArchestrA 系统管理控制台，简称 SMC）中，它随同 InTouch HMI 一起安装。

要查看 Log Viewer 中记录的值

- 1 单击开始，指向程序，指向 **Wonderware**，然后单击 **ArchestrA 系统管理控制台**。此时出现 ArchestrA 系统管理控制台（ArchestrA 系统管理控制台）。



- 2 在左侧窗格中，展开 **Log Viewer**，展开 **Default Group**（缺省组），然后单击 **Local**（本地）。此时详细信息窗格中出现 Log Viewer 消息。



- 3 找到 LogMessage() 函数所记录的值。

备注 如果在远程 InTouch HMI 节点上调试脚本，则必须将节点名添加到 Log Viewer 中的节点组，然后查看该节点的 Log Viewer 消息。

索引

符号

\$AccessLevel 系统标记 145
\$ChangePassword 系统标记 144
\$ConfigureUsers 系统标记 145
\$Date 系统标记 113
\$DateString 系统标记 114
\$DateTime 系统标记 113
\$Day 系统标记 110
\$Hour 系统标记 111
\$InactivityTimeout 系统标记 145
\$InactivityWarning 系统标记 145
\$LogicRunning 系统标记 37
\$Minute 系统标记 111
\$Month 系统标记 110
\$Msec 系统标记 112
\$OperatorDomainEntered 系统标记 144
\$OperatorEntered 系统标记 144
\$PasswordEntered 系统标记 144
\$Second 系统标记 112
\$Time 系统标记 112
\$TimeString 系统标记 114
\$Year 系统标记 110

A

Abs() 函数 70
ActivateApp() 函数 122
ActiveX
 创建事件脚本 169
 调用方法 166
 复用事件脚本 171
 将属性链接到标记 168
 控件方法 165
 控件浏览器 166
 控件属性 167
 控件属性读取和写入数据 167
 在脚本中插入方法或属性 19
ActiveX 事件脚本 169–173
 编辑 36
 触发器 22
 创建 170
 打开 16
 导入 173
 复用 171
 配置 34
 删除 36
 自引用 172
AddPermission() 函数 145
ArcCos() 函数 74

ArcSin() 函数 74
ArcTan() 函数 75
ASCII 码转换 82
AttemptInvisibleLogon() 函数 144
安全性相关脚本 144

B

保存更改 17
比较字符串 88
编辑脚本 15–19
编辑器 13
变量
 命名冲突 61
 声明 60
 使用 60
表达式示例 53
标记命名冲突 61
标记引用 41
播放声音文件 146
不正确嵌套的示例 56
不正确嵌套示例 56

C

ChangePassword() 函数 144
ChangeWindowColor() 函数 103
Cos() 函数 74
CSV 文件函数 133
查看日志消息 178
插入
 代码元素 18
 点域 18
插入标记名 18
插入点域 18
插入函数 18
查找与 / 或替换文本 18
程序分支 54
程序循环 56
处理文件 129–137
创建
 ActiveX 事件脚本 169, 172
 脚本 15
 OLE 对象 154
 用户界面对话框 161
 自定义脚本 63

窗口
 打开 98
 打印 104
 更改颜色 103
 关于打印的建议 105
 移动与调整大小 101
 隐藏 102
 与其它应用程序交互 120

窗口脚本
 触发器 22
 打开 16
 配置 24
窗口状态 97
从字符串中删除空格 81

D

DateTimeGMT() 函数 113
打开
 InTouch 窗口 98
 脚本进行编辑 16
 Windows 日期 / 时间属性面板 163
DText() 函数 96
打印窗口 104
打印脚本 20
打印历史趋势 108
导入 ActiveX 事件脚本 173
登录与注销 144
调用
 ActiveX 控件方法 165
 标准函数 42
 OLE 对象的方法 157
 QuickFunction 66
 自定义函数 44
调用 QuickFunction 44
调用自定义函数 44
定期执行脚本 22
动画显示链接强制更新 69
动作脚本
 触发器 22
 打开 16
 配置 31
 删除 33
 条件类型列表 32
读取和写入

CSV 数据 133

文本数据 136

注册表 164

对数 76

多个触发器 22

E

Exp() 函数 76

F

FileCopy() 函数 129

FileDelete() 函数 131

FileMove() 函数 132

FileReadFields() 函数 133

FileReadMessage() 函数 136

FileWriteFields() 函数 135

FileWriteMessage() 函数 137

FOR 循环 56

强制结束 58

返回

关于字符串的信息 86

Pi 的值 75

字符串的某些部分 78

放弃更改 17

访问 OLE 对象的属性 156

分支

IF-THEN-ELSE 54

嵌套 55

无效示例 55, 56

分支结构 54

复用 ActiveX 事件脚本 171

赋值语句与运算符 45, 45–53

复制、剪切及粘贴文本 17

G

GetAccountStatus() 函数 145

GetNodeName() 函数 138

GetPropertyD() 函数 147

GetPropertyI() 函数 149

GetPropertyM() 函数 150

更改

窗口的颜色 103

口令 144

离散 ActiveX 控件缺省属性值 167

字符串大小写 80

更新动画显示链接 69

关闭、最小化或最大化 Windows 应用程序 125

管理安全性及其它信息 145

管理文件 129

关于打印窗口的建议 105

H

Hide() 函数 102

HideSelf() 函数 102

函数

Abs() 函数 70

ActivateApp() 函数 122

AddPermission() 函数 145

ArcCos() 函数 74

ArcSin() 函数 74

ArcTan() 函数 75

AttemptInvisibleLogon() 函数 144

ChangePassword() 函数 144

ChangeWindowColor() 函数 103

Cos() 函数 74

DateTimeGMT() 函数 113

DText() 函数 96

调用语法 42

定义 12

Exp() 函数 76

FileCopy() 函数 129

FileDelete() 函数 131

FileMove() 函数 132

FileReadFields() 函数 133

FileReadMessage() 函数 136

FileWriteFields() 函数 135

FileWriteMessage() 函数 137

GetAccountStatus() 函数 145

GetNodeName() 函数 138

GetPropertyD() 函数 147

GetPropertyI() 函数 149

GetPropertyM() 函数 150

Hide() 函数 102

HideSelf() 函数 102

InfoAppActive() 函数 121

InfoAppTitle() 函数 121

InfoDisk() 函数 139

InfoFile() 函数 140
InfoInTouchAppDir() 函数 142
InfoResources() 函数 141
Int() 函数 71
InTouchVersion() 函数 143
InvisibleVerifyCredentials() 函数 145
IsAnyAsyncFunctionBusy() 函数 67
IsAssignedRole() 函数 145
Log() 函数 76
LogMessage() 函数 177
LogN() 函数 77
Logoff() 函数 144
LogonCurrentUser() 函数 144
OLE_CreateObject() 函数 154
OLE_GetLastObjectError() 函数 159
OLE_GetLastObjectErrorMessage() 函数 159
OLE_IsObjectValid() 函数 154
OLE_ReleaseObject() 函数 155
OLE_ResetObjectError() 函数 159
OLE_ShowMessageOnObjectError() 函数 160
OpenWindowsList() 函数 97
PlaySound() 函数 146
PostLogonDialog() 函数 144
PrintHT() 函数 108
PrintScreen() 函数 107
PrintWindow() 函数 106
QueryGroupMembership() 函数 145
Round() 函数 71
SendKeys 123
SetPropertyD() 函数 148
SetPropertyI() 函数 150
SetPropertyM() 函数 151
SetWindowPrinter() 函数 104
Sgn() 函数 72
Show() 函数 98
ShowAt() 函数 99
ShowHome() 函数 99
ShowTopLeftAt() 函数 100
Sin() 函数 73
Sqrt() 函数 77
StartApp 120
StringASCII() 函数 83

StringChar() 函数 82
StringCompare() 函数 88
StringCompareEncrypted() 函数 90
StringCompareNoCase() 函数 89
StringFromIntg() 函数 92
StringFromReal() 函数 93
StringFromTime() 函数 116
StringFromTimeLocal() 函数 118
StringInString() 函数 84
StringLeft() 函数 78
StringLen() 函数 86
StringLower() 函数 80
StringMid() 函数 79
StringReplace() 函数 85
StringRight() 函数 79
StringSpace() 函数 82
StringTest() 函数 86
StringToIntg() 函数 94
StringToReal() 函数 95
StringTrim() 函数 81
StringUpper() 函数 80
三角 73
Tan() 函数 75
Text() 函数 91
Trunc() 函数 72
WindowState() 函数 97
UTCDateTime() 函数 115
WWControl() 函数 125
WWExecute() 函数 126
wwIsDaylightSavings() 函数 119
WWMoveWindow() 函数 101
WWPoke() 函数 128
WWRequest() 函数 127
wwStringFromTime() 函数 117
传递参数 43
获取与设置向导属性 147

I
IF-THEN-ELSE 分支 54
InfoAppActive() 函数 121
InfoAppTitle() 函数 121
InfoDisk() 函数 139
InfoFile() 函数 140

InfoInTouchAppDir() 函数 142

InfoResources() 函数 141

Int() 函数 71

InTouchVersion() 函数 143

InvisibleVerifyCredentials() 函数 145

IsAnyAsyncFunctionBusy() 函数 67

IsAssignedRole() 函数 145

J

激活正在运行的 Windows 应用程序 122

记录消息 175–178

计算 70

对数 76

Pi 75

平方根 77

技术支持 10

检查

窗口是打开、关闭或是否不存在 97

是否有任何异步 QuickFunction 正在运行 67

夏令时状态 119

指定的应用程序是否正在运行 121

键脚本

触发器 22

配置 26

删除 27

检索

磁盘空间信息 139

InTouch 相关信息 142

计算机的节点名 138

数值日期与时间信息 109

Windows 环境的有关信息 141

文件或目录的有关信息 140

系统相关信息 138

应用程序标题 121

字符串日期与时间信息 114

检索 InTouch 相关信息 142–143

检索系统相关信息 138–142

将参数传递给函数 43

将多个指针指定给相同的 OLE 对象 158

将离散值转换为字符串 96

将模拟键击发送到应用程序 123

将日期与时间转换为字符串 116

将整数转换为字符串 92

将字符串转换为整数 94

脚本

保存更改 17

播放声音文件 146

插入 ActiveX 方法或属性 19

插入标记名 18

插入窗口名 19

插入代码元素 18

插入点域 18

插入关键字或运算符 19

插入函数 18

查找与 / 或替换文本 18

打开 16

打印 20

定期执行 22

定义 12

放弃更改 17

复制、剪切及粘贴文本 17

类型 22

使用 DDE 126

验证 19

语法规则 40

语句 40

暂停执行 37

脚本编辑器 13

脚本触发器 21–22

脚本触发器的类型 22

脚本函数的帮助 19

脚本示例

创建用户界面对话框 161

从字符串中提取实数 95

从字符串中提取整数 94

监视异步函数 67

嵌套循环 59

声明局部变量 60

使用循环插入数据库记录 58

使用循环来初始化标记 59

用于删除空格的循环 81

脚本语言概述 39

脚本中的注释 40

局部变量 60–61

命名冲突 61

声明 60

使用 60

K

口令的设置与更改 144

L

Log Viewer 178

Log() 函数 76

LogMessage() 函数 176, 177

LogN() 函数 77

Logoff() 函数 144

LogonCurrentUser() 函数 144

历史趋势打印 108

M

MEM OLE 13

命名冲突 61

目录信息检索 140

O

OLE

产生随机数 161

创建用户界面对话框 161

错误 159

打开 Windows 日期 / 时间属性面板 163

对注册表进行读取和写入 164

复位上一个错误 159

统计错误消息数 160

显示或隐藏错误消息 160

最小化窗口 164

OLE 对象

创建 154

错误 159

调用方法 157

读取属性 156

访问属性 156

将多个指针指定给 158

释放 155

属性与方法 156

写入属性 157

有效性 154

OLE_CreateObject() 函数 154

OLE_GetLastObjectError() 函数 159

OLE_GetLastObjectErrorMessage() 函数 159

OLE_IsObjectValid() 函数 154

OLE_ReleaseObject() 函数 155

OLE_ResetObjectError() 函数 159

OLE_ShowMessageOnObjectError() 函数 160

OpenWindowsList() 函数 97

P

Pi 75

PlaySound() 函数 146

PostLogonDialog() 函数 144

PrintHT() 函数 108

PrintScreen() 函数 107

PrintWindow() 函数 106

排解 OLE 错误 159–160

配置

ActiveX 控件属性以读取和写入数据 167

ActiveX 事件脚本 34

窗口脚本 24

动作脚本 31

键脚本 26

QuickFunction 64

数据改变脚本 30

条件脚本 28

应用程序脚本 23–24

平方根 77

Q

启动 Windows 应用程序 120

其它脚本 146

QueryGroupMembership() 函数 145

QuickFunction 63

创建 64

创建异步 66

调用 44, 66

定义 12, 63

检查异步 67

配置 64

删除 65

停止异步 68

修改 65

异步限制 66

传递参数给 44

QuickScript

关于语言 12

将消息记录到 LogViewer 175
设置检查点 176
疑难排解 175
QuickScript 疑难排解 175–178
嵌套条件结构 55
强制结束循环 58

R

Round() 函数 71
日期与时间 109–119
日期与时间的系统标记 109, 114
日期与时间系统标记 109
日期与时间作为字符串检索 114

S

SendKeys 123
SetPropertyD() 函数 148
SetPropertyI() 函数 150
SetPropertyM() 函数 151
SetWindowPrinter() 函数 104
Sgn() 函数 72
Show() 函数 98
ShowAt() 函数 99
ShowHome() 函数 99
ShowTopLeftAt() 函数 100
Sin() 函数 73
Sqrt() 函数 77
StartApp 120
StringASCII() 函数 83
StringChar() 函数 82
StringCompare() 函数 88
StringCompareEncrypted() 函数 90
StringCompareNoCase() 函数 89
StringFromIntg() 函数 92
StringFromReal() 函数 93
StringFromTime() 函数 116
StringFromTimeLocal() 函数 118
StringInString() 函数 84
StringLeft() 函数 78
StringLen() 函数 86
StringLower() 函数 80
StringMid() 函数 79
StringReplace() 函数 85
StringRight() 函数 79

StringSpace() 函数 82
StringTest() 函数 86
StringToIntg() 函数 94
StringToReal() 函数 95
StringTrim() 函数 81
StringUpper() 函数 80
三角函数 73
删除脚本 20
设置口令 144
设置运算符的求值顺序 51
声明局部变量 60
释放 OLE 对象 155
事件脚本
 创建 169
 导入 173
 复用 171
 自引用 172
实数值
 从字符串转换 95
 转换为字符串 93
使用 DDE 执行命令与交换数据 126
使用 OLE 产生随机数 161
使用程序循环 56–59
使用空格设置字符串格式 82
使用条件程序分支 54
数据改变脚本
 触发器 22
 配置 30
 删除 30
数据类型转换 52
数据值 41
属性列表 147–151
数学计算 70–77
搜索与替换字符串中的文本 83
缩进脚本语句 40

T

Tan() 函数 75
Text() 函数 91
Trunc() 函数 72
条件程序分支 54–56
条件脚本
 触发器 22
 配置 28

删除 29
停止 QuickFunction 68

U

UTCDateTime() 函数 115

W

Windows

发送键击 123
关闭、最小化或最大化 125
激活 122
激活应用程序 122
检索环境信息 141
启动 120
应用程序标题 121
正在运行 121

WindowState() 函数 97

WWControl() 函数 125

WWExecute() 函数 126

wwIsDaylightSavings() 函数 119

WWMoveWindow() 函数 101

WWPoke() 函数 128

WWRequest() 函数 127

wwStringFromTime() 函数 117

文档惯例 9

文件

CSV 函数 133
检索目录信息 140
文本函数 136
文件操作 129

X

夏令时 119

显示打开的窗口的列表 97

限制

异步 QuickFunction 66
应用程序脚本 24

向导

获取与设置属性 147
属性函数列表 147

循环 59

对其它进程的影响 58

FOR 56

强制结束 58

示例 59
使用 56
执行的时间限制 59

Y

验证 OLE 对象 154

验证脚本 19

异步 QuickFunction

检查 67
停止 68
限制 66

移动窗口与调整大小 101

隐藏 InTouch 窗口 102

隐式数据类型转换 52

应用程序脚本

触发器 22
打开 16
配置 23
限制 24

与安全性相关的脚本 144

语法

标记引用 41
规则 40
数据值 41
缩进 40
验证 19, 41
语句 40
值表达式 41
注释 40
子程序 40

语法规则 40–41

语句 40

与其它应用程序交互 120–128

运算符

AND 48, 49
比较 51
乘法 46
除法 47
串联 46
加法 46
减法 46
逻辑非 NOT 50
逻辑或 50
逻辑与 49

- 幂 47
- 模除 MOD 47
- NOT 50
- OR 49, 50
- 求值顺序 51
- 取补 47
- 取反 46
- SHL 47
- SHR 47
- 位非 49
- 位或 49
- 位与 48
- XOR 49
- 右移位 47
- 支持的 45
- 左移位 47
- 运行异步 QuickFunction 66–68
- Z**
 - 在动画显示链接中强制更新 69
 - 暂停脚本执行 37
 - 正确的语法 19
 - 指定与配置用户 145
 - 传递参数给 QuickFunction 44
 - 转换
 - ASCII 码 82
 - 离散值转换为字符串 96
 - 日期与时间转换为字符串 116
 - 整数或实数转换为字符串 91
 - 整数值转换为字符串 92
 - 字符 82
 - 字符串转换为实数值 95
 - 字符串转换为整数值 94
 - 转换数据类型 90–96
 - 子程序 40
 - 字符串
 - 比较 88
 - 从中删除空格 81
 - 返回部分 78
 - 返回有关信息 86
 - 更改大小写 80
 - 实数值转换为 93
 - 使用空格设置格式 82
 - 替换文本 83
 - 整数或实数转换为 91
 - 转换离散值 96
 - 转换日期与时间 116
 - 转换为实数值 95
 - 转换为整数值 94
 - 转换整数值 92
 - 字符串运算 78–90
 - 自引用 ActiveX 事件脚本 172
 - 最小化窗口 164

