SIEMENS

WinCC

Руководство по конфигурации

Том 2

Данное руководство является частью пакета документации
с заказным номером:
6AV6392–1CA05–0AB0
C79000–G8276–C158–01

Выпуск: Сентябрь 1999

C79000–G8276–C158

Siemens Aktiengesellschaft

Содержание

Предисловие

Цель руководства

Данное руководство знакомит Вас с существующими опциями конфигурации WinCC, в частности с разделами:

Запуск примеров

Конфигурация тегов/переменных

Конфигурация кадров

Редакторы WinCC

Данное руководство можно получить как в печатном, так и в электронном виде.

Оглавление и предметный указатель помогут вам быстро найти необходимую информацию. В электронной версии предусмотрены расширенные возможности поиска.

Требования к пользователям данного руководства

Знание основ WinCC, например, на основе руководства для начинающих “Getting Started“ или из практического опыта конфигурирования WinCC.

Дополнительная поддержка

По техническим вопросам обращайтесь в представительство компании Siemens в Вашем регионе.

Также вы можете воспользоваться нашей горячей линией:

+49 (911) 895–7000 (Fax –7001)

Информация о продуктах SIMATIC

Актуальную информацию о продуктах SIMATIC можно найти в каталоге CA01. Данный каталог расположен в Internet по адресу:

http://www.ad.siemens.de/ca01online/

Кроме того, служба поддержки пользователей SIMATIC предоставляет клиентам текущую информацию и загружаемые программы. Подборка часто задаваемых вопросов находится в Internet по адресу:

http://www.ad.siemens.de/support/html_00/index.shtml

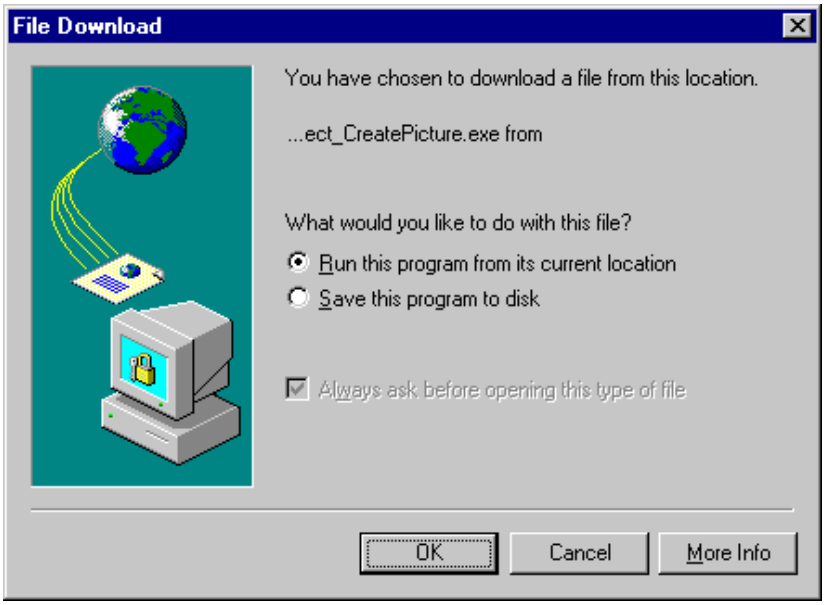
1 Запуск примеров

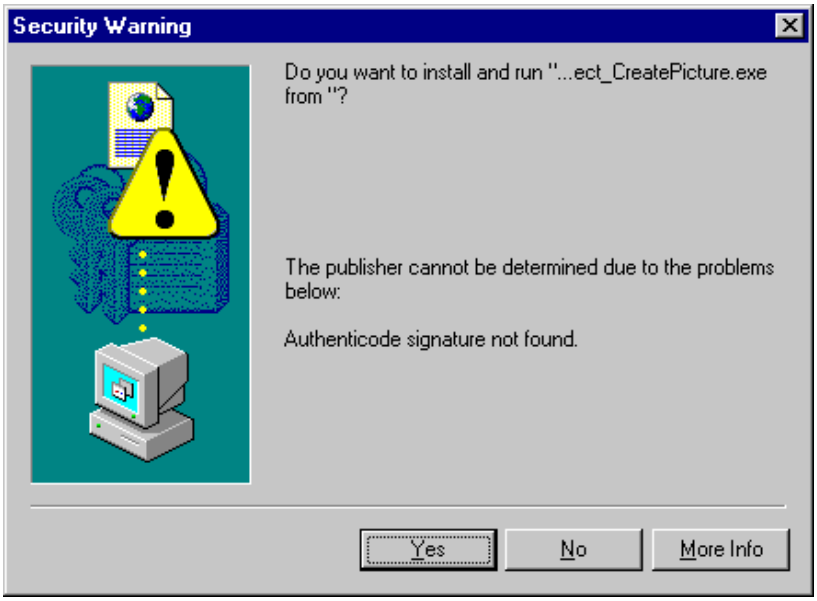
В этом разделе руководства мы опишем приемы работы в WinCC на примере типовых проектов. Принимая во внимание широкий спектр потенциальных приложений WinCC, описанные ниже проекты должны рассматриваться только как демонстрация возможностей WinCC.

Проекты WinCC, созданные для этого раздела руководства, могут быть скопированы непосредственно из онлайнового документа на ваш жесткий диск. По умолчанию они будут записаны в папку *C:\Configuration_Manual*. Действия, которые необходимо выполнить для запуска проектов WinCC, описаны в приведенной ниже таблице.

1.1 Загрузка примеров

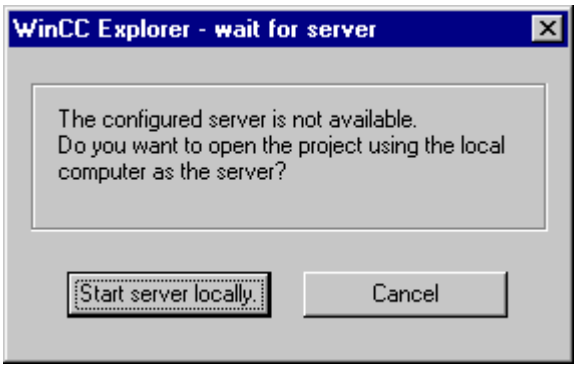
Загрузка примеров

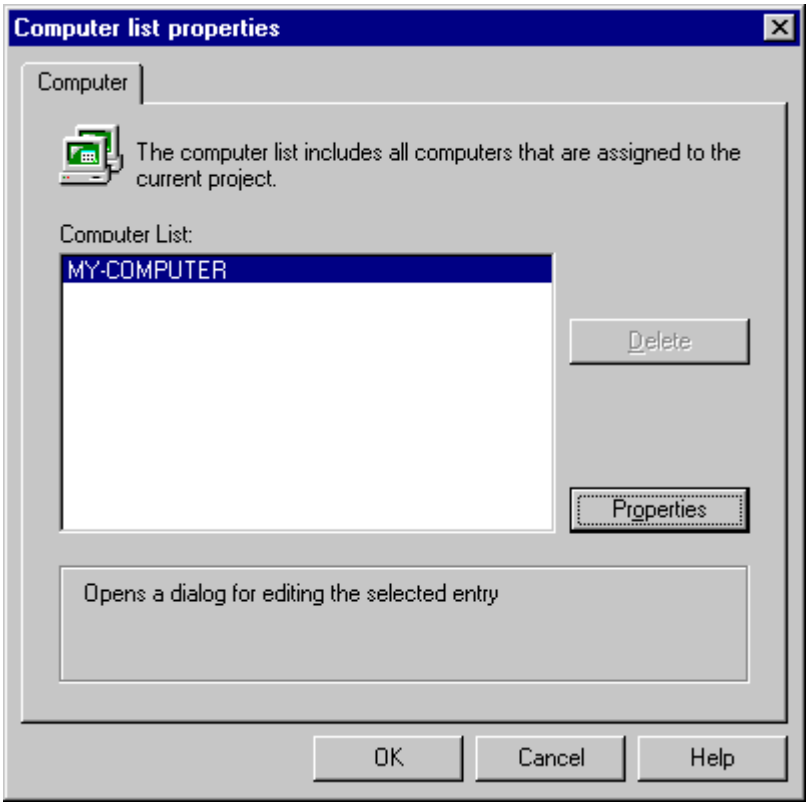
Шаг	Процедура: Загрузка примеров
1	Загрузка необходимого проекта. Выполняется из интерактивного документа посредством щелчка (мыши) на следующей иконке: Имя проекта
2	Появится диалоговое окно *Download File (Загрузка файла)*. Выберите пункт *Run this program from its current location (Запустить программу из текущего местонахождения)*. Подтвердите решение нажатием на *OK*. 

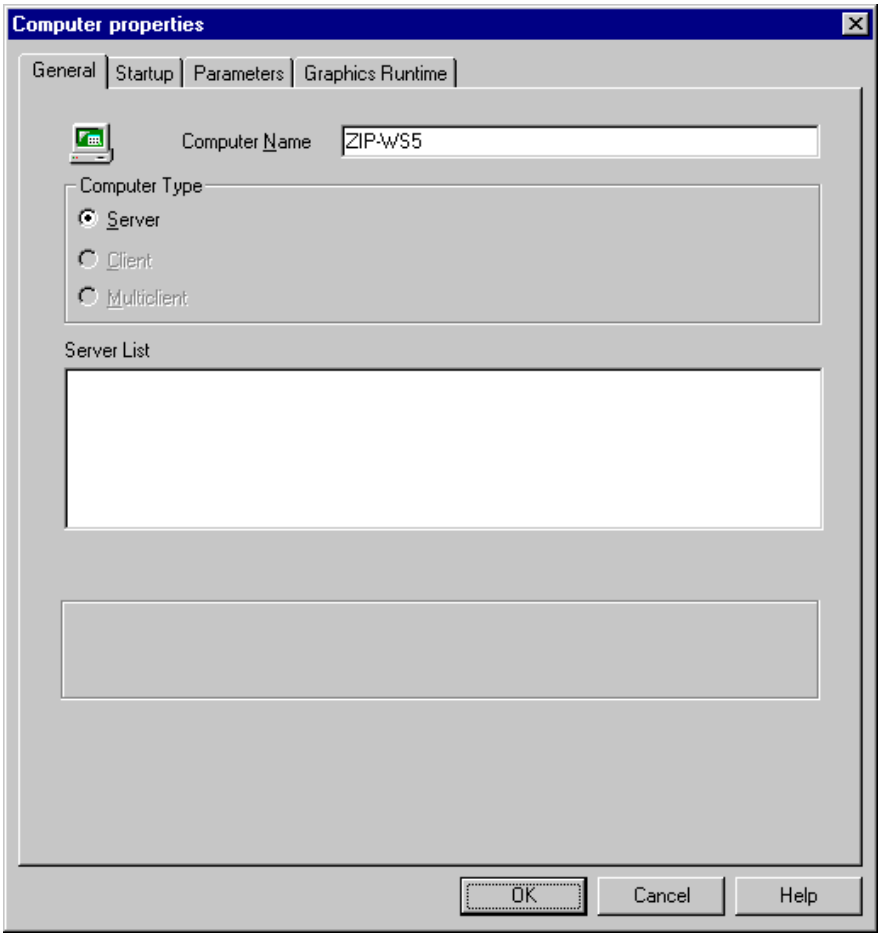
Шаг	Процедура: Загрузка примеров
3	Появится диалоговое окно *Security Warning (Предупреждение безопасности)*. Подтвердите этот диалог нажатием на *Yes (Да)*. 
4	Запустится диалоговое окно *WinZip Self–Extractor (Самораспаковывающийся архив WinZip)*. В нем вы можете указать папку, в которую нужно распаковать архив. По умолчанию проекты будут распакованы в папку *C:\Configuration_Manual*. Процесс распаковки запускается нажатием на кнопку *Unzip (Распаковать)*. 
5	После завершения распаковки будет выведено окно с сообщением об успешном завершении процесса. Нажмите на кнопку *OK*. Для завершения диалога *WinZip Self–Extractor (Самораспаковывающийся архив WinZip)* нажмите кнопку *Close (Закрыть)*. 

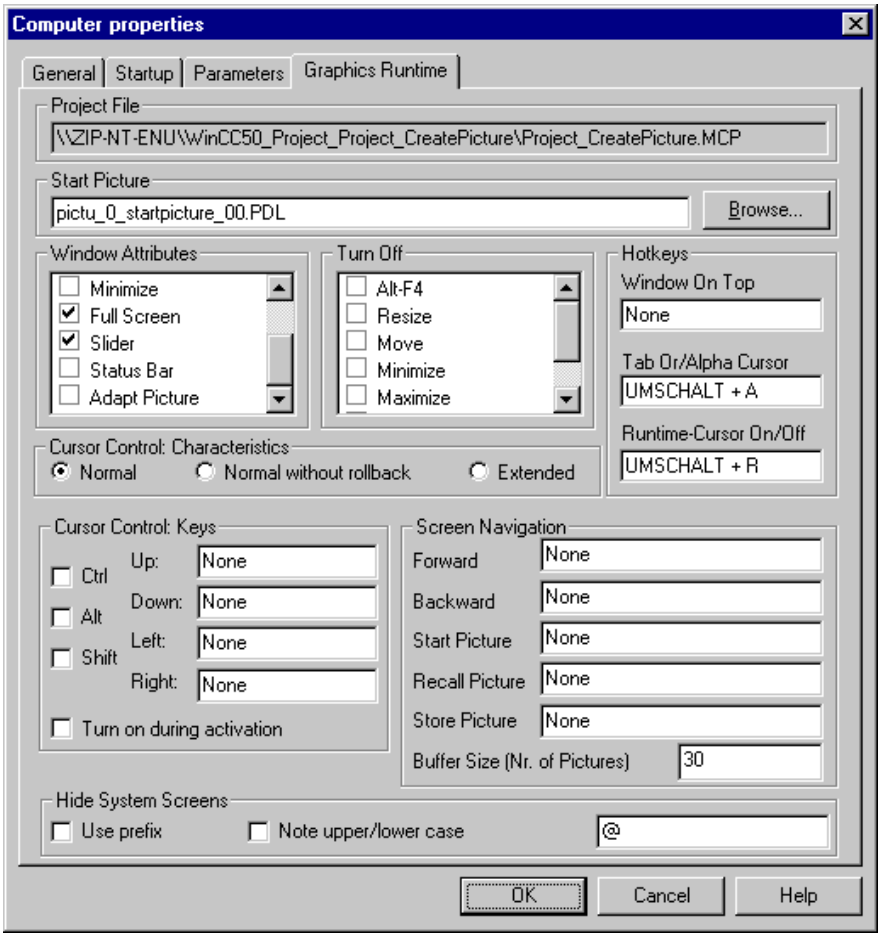
1.2 Запуск примеров (однопользовательские проекты)

Запуск примеров (однопользовательские проекты)

Шаг	Процедура: Запуск примеров (однопользовательские проекты)
1	Откройте *WinCC Explorer (Проводник WinCC)*. Откройте проект–пример, который был распакован. Будет выведено диалоговое окно с сообщением о том, что сконфигурированный сервер недоступен. Нажатием на *Start Local Server (Запуск локального сервера)*, этот проект WinCC все же можно открыть. 
2	Для работы с проектом необходимо в качестве локального сервера указать название вашего компьютера. Это можно сделать в *WinCC Explorer (Проводнике WinCC)*, щелкнув R (правой кнопкой мыши) на пункт *Computer (Компьютер)* и выбрав пункт *Properties (Свойства)* во всплывающем меню. 

Шаг	Процедура: Запуск примеров (однопользовательские проекты)
3	Появится диалоговое окно *Computer List Properties (Свойства списка компьютеров)*. Этот список включает в себя все компьютеры, имеющие отношение к проекту. Кнопка *Properties (Свойства)* открывает доступ к свойствам компьютера. 

Шаг	Процедура: Запуск примеров (однопользовательские проекты)
4	Появится диалоговое окно свойств компьютера. На закладке *General Information (Общие сведения)* введите имя локального компьютера. 

Шаг	Процедура: Запуск примеров (однопользовательские проекты)
5	На закладке *Graphics–Runtime (Графика времени исполнения)* убедитесь, что все установки верны. Помимо прочего, проверьте, указано ли стартовое изображение. Если разрешающая способность экрана меньше 1024 x 768, флажки *Full Screen (Полноэкранный режим)* и *Scroll Bars (Полосы прокрутки)* в атрибутах окна должны быть выставлены. Выйдите из диалога, нажав на *OK*. Выйдите из диалога свойств компьютера, также нажав на *OK*. 
6	Перед запуском проекта его необходимо перезагрузить. Закройте проект, используя команду *File (Файл)* → *Close (Закрыть)*, и откройте его снова.

Примечание:

Описанные действия относятся к однопользовательским проектам. Они же могут быть выполнены и для многопользовательских проектов, описанных в руководстве, однако, для таких проектов требуются некоторые дополнительные действия. Они будут описаны в последующих примерах.

2 Конфигурация тегов/переменных (Project_TagHandling)

Проект WinCC, созданный в этой главе, можно скопировать непосредственно из online–документа на ваш жесткий диск. По умолчанию он будет записан в папку *C:\Configuration_Manual*.


Project_TagHandling

В этом проекте вы найдете указания, которые облегчат работу с тегами/переменными в WinCC. В целом, в WinCC имеется три различных типа тегов. Это внутренние теги, не подключенные к драйверу процесса, теги WinCC (также называемые внешними тегами), связанные с драйвером процесса, и переменные *Cu* в коде программ и функций. Примеры, имеющие отношение к проекту *Project_TagHandling*, в основном ориентированы на работу с внутренними тегами. В целом, обработка этих тегов незначительно отличается от обработки тегов WinCC.

Примеры для этого раздела приведены в проекте WinCC *Project_TagHandling*. Вид его начальной страницы показан ниже.

2.1 Создание, группировка и перемещение тегов

В *WinCC Explorer (Проводнике WinCC)* теги создаются в *Tag Management (Менеджере тегов)*. Существует различие между не связанными с драйвером процесса тегами, называемыми *Internal Tags (Внутренними тегами)*, и тегами, связанными с драйвером процесса, называемыми *WinCC Tags (Тегами WinCC)*, или *External Tags (Внешними тегами)*. Для внутренних тегов нет ограничений на их количество. В то же время максимально возможное количество *тегов WinCC* зависит от приобретенной лицензии.

Группы Тегов и Теги

При обработке большого количества данных и, следовательно, большого количества тегов, рекомендуется объединять эти теги в группы. Только таким образом можно отслеживать правильность конфигурации в крупномасштабных проектах. Тем не менее, группы тегов никак не гарантируют уникальность используемых тегов. Это делается исключительно посредством имен тегов.

Шаг	**Процедура: Группы тегов и теги**
1	Создание группы тегов для *внутренних тегов* выполняется в *менеджере тегов* посредством нажатия R (правой кнопки мыши) на пункте *Internal Tags (Внутренние теги)* и выбора элемента *New Group... (Новая группа...)* во всплывающем меню. 
2	В появившемся диалоговом окне группе необходимо присвоить имя. После этого в *проводнике WinCC* будет отображено это имя вместе с иконкой для новой группы. В типовом проекте *Project_TagHandling* разделение на группы было произведено в соответствии с рассматриваемыми главами.

Шаг	Процедура: Группы Тегов и Теги
3	Создание тега в группе тегов осуществляется нажатием на R (правую кнопку мыши) на группе и выбором пункта *New Tag…(Новый Тег…)* во всплывающем меню. 
4	В появившемся диалоговом окне, нужно задать имя тега на закладке *General Information (Общие сведения)*. Ниже, в списке выбора, следует указать подходящий тип данных *(Data Type)*. Для внутренних тегов указывать адрес *(Address)* не нужно.

Примечание:

При запущенном проекте в проводнике WinCC используя всплывающую подсказку можно посмотреть текущее значение и статус тега в кадре процесса.

Перемещение Тегов

Шаг	Процедура: Перемещение Тегов
1	В *менеджере тегов* тег перемещается нажатием на нем R (правой кнопки мыши), и выбором пункта *Cut (Вырезать)* во всплывающем меню. После этого выбирается искомая группа тегов. Тег вставляется в нее посредством нажатия R (правой кнопки мыши), и выбора команды *Paste (Вставить)* во всплывающем меню. Данная процедура может быть применена к нескольким тегам одновременно.

Примечание:

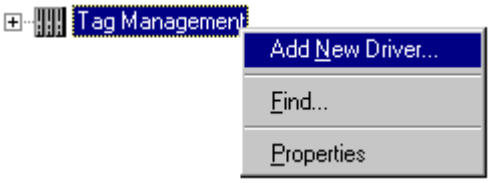
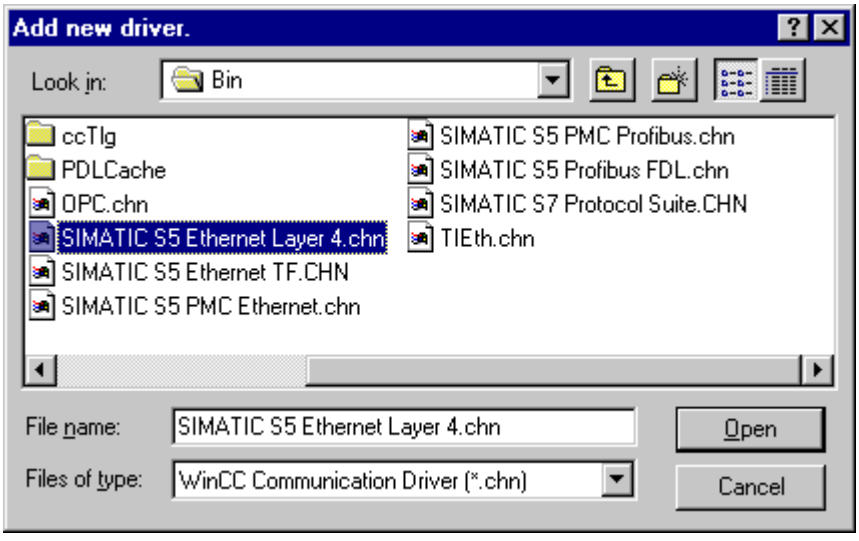
При перемещении или удалении тегов в *проводнике WinCC*, проект не должен находиться в режиме исполнения.

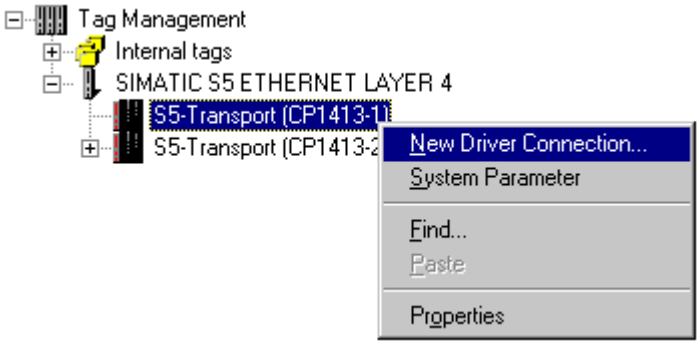
Если вам требуется большое количество одноименных, последовательно пронумерованных тегов, вам необходимо создать только один тег такого типа.

Этот тег можно скопировать в системный буфер обмена, нажав на R (правую кнопку мыши) и выбрав *Copy (Копировать)* во всплывающем меню — тег можно будет вставлять любое количество раз. Теги будут пронумерованы автоматически по возрастанию. Вы должны иметь в виду эту возможность при задании имен тегов.

Теги WinCC

Для создания *тегов WinCC* в *менеджере тегов* предварительно необходимо установить связь с PLC. Однако для этого не требуется установка дополнительных аппаратных средств. Достаточно установить драйвер связи и настроить требуемое подключение.

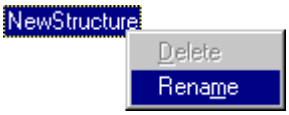
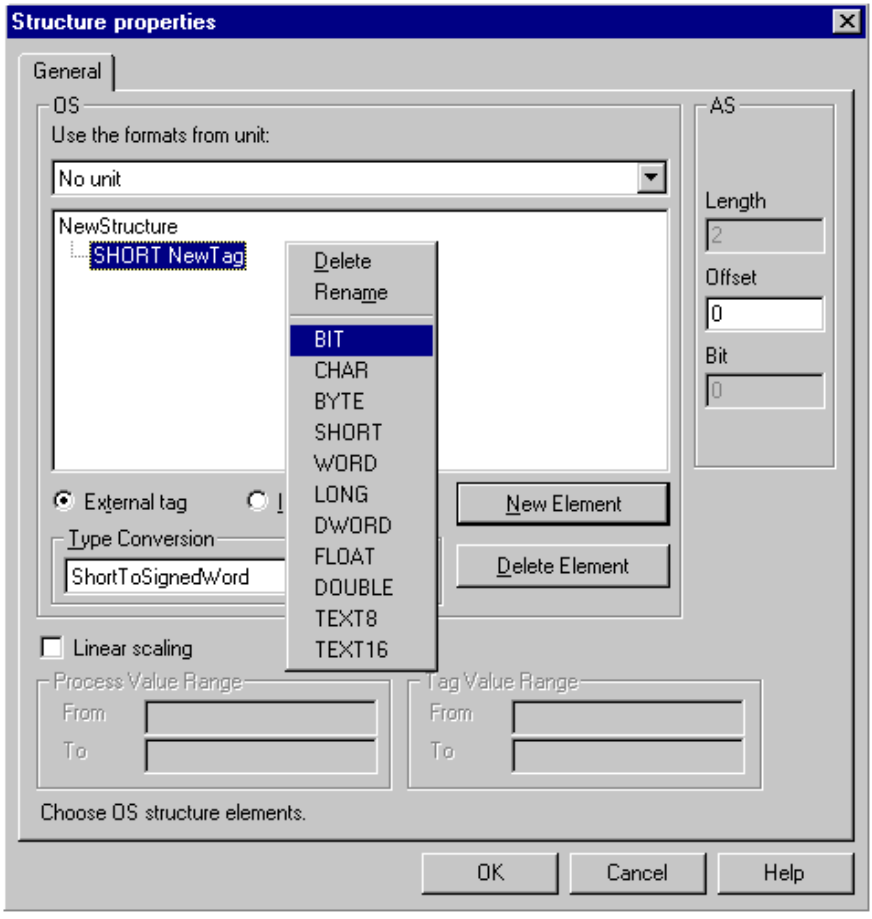
Шаг	Процедура: Теги WinCC
1	Установка нового драйвера связи. Выполняется нажатием на R (правую кнопку мыши) в *менеджере тегов* и выбором пункта *Add New Driver…(Добавить новый драйвер…)* во всплывающем меню. 
2	В появившемся диалоговом окне выберите необходимый драйвер. Нажатием на кнопку *Open (Открыть)* драйвер вставляется в проект WinCC. В *менеджере тегов проводника WinCC* в дополнение к *внутренним тегам* появится новый драйвер. 

Шаг	Процедура: Теги WinCC
3	По DD (двойному щелчку мыши) на новом драйвере, будут показаны один или несколько подпунктов, называемых *Channel Units (Модули Канала)*. Создание подключения. Выполняется нажатием на R (правую кнопку мыши) на элементе *Channel Unit (Модуль Канала)* и выбором пункта *New Driver Connection (Новый драйвер соединения)* во всплывающем меню. 
4	В появившемся диалоговом окне на закладке *General Information (Общие сведения)* присвойте соединению имя. Параметры подключения можно настроить, нажав на кнопку *Properties (Свойства)*.
5	По нажатию R (правой кнопкой мыши) на вновь созданном подключении, теги и группы тегов можно добавить указанным выше способом.
6	При создании *тегов WinCC* помимо установок, выполняемых для *внутренних тегов*, необходимо также задать адрес и формат преобразования. Адресами являются адреса тегов в PLC.

Структурные теги

Структурные теги используются для группировки большого количества различных тегов, образующих логическую единицу. При этом к самому тегу и к его элементам можно обращаться, используя одно и то же имя.
Структурный тег состоит из ряда отдельных тегов, которые могут представлять различные типы данных.

Шаг	Процедура: Структурный тег
1	Новая структура создается нажатием на R (правую кнопку мыши) на элементе *Structure Type (Структурный тип)* и выбором пункта *New Structure Type (Новый структурный тип)* во всплывающем меню. 

Шаг	Процедура: Структурный тег
2	В появившемся диалоговом окне присвойте структуре новое имя нажатием R (правой кнопки мыши) на элементе *NewStructure (Новая структура)* и выбором пункта *Rename (Переименовать)* во всплывающем меню. 
3	Новый элемент структуры может быть добавлен по кнопке *New Element (Новый Элемент)*.
4	Нажатием на R (правую кнопку мыши) на вновь созданном элементе можно задать его имя и тип данных. Для каждого элемента структуры вы должны указать, является ли он *internal (внутренним)* или *external tag (внешним тегом)*. После нажатия на кнопку *OK* конфигурирование завершается и создается структурный тип. 

Примечание:

Однажды созданный структурный тип не может быть модифицирован в дальнейшем. Для этого необходимо повторно определить весь структурный тип.

Структура создается так же, как и все другие типы тегов, однако в качестве типа данных следует указать вновь созданный структурный тип. Имена отдельных элементов структуры тегов образуются из имени структуры, присваиваемого ей при создании, и имени элемента, задаваемого при создании типа. Эти две части разделяются в названии точкой.

Name	Type	Parameters	Last change
STUi_varia_str_00.aktivated	Binary Tag	Internal tag	07/10/97 02:26:14
STUi_varia_str_00.open	Binary Tag	Internal tag	07/10/97 02:26:14
STUi_varia_str_00.closed	Binary Tag	Internal tag	08/21/97 03:45:32
STUi_varia_str_00.error	Binary Tag	Internal tag	07/10/97 02:26:14

2.2 Инкрементирование, декрементирование, дискретное изменение

В режиме исполнения примеры, имеющие отношение к этой теме, доступны в проекте *Project_TagHandling* по нажатию (мышью) на *кнопке*, показанной выше. Примеры приведены в кадрах в *varia_3_chapter_01.pdl* и *varia_3_chapter_01a.pdl*.

Определения

Инкрементирование — это увеличение значения тега на фиксированное или переменное значение.

Декрементирование — это уменьшение значения тега на фиксированное или переменное значение.

Дискретное изменение — это выполнение определенного действия, например связанного с нажатием кнопки. В случае дискретных сигналов, как правило, осуществляется управление устройством. В случае аналоговых величин, производится пошаговое изменение заданного значения.

2.2.1 Дискретное изменение — изменение значения уставки (example 01)

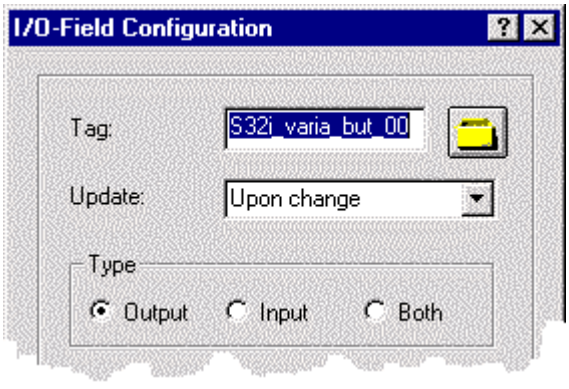
Постановка задачи

Дискретное изменение параметра необходимо производить при помощи мыши. Изменение значения уставки на фиксированную величину должно производиться по нажатию кнопки. Изменение значения должно быть ограничено установленными пределами. Изменения должны быть реализованы локально в рамках экранной формы.

Концепция реализации

Для реализации используются два объекта *Windows Object (Объект Windows)* → *Buttons (Кнопка)*, с помощью которых уставка изменяется по управляющему событию. При нажатии на *кнопку* (мышью) значение *внутреннего тега* изменяется на единицу инкремента. Единица инкремента задается заранее и не может быть изменена в режиме исполнения. Изменение уставки реализовано в *C–Action (Процедуре Си)*.
Изменение значения уставки отображается в *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. Выходное значение *поля ввода/вывода* связывается с *внутренним тегом*.

Реализация в графическом редакторе

Шаг	Процедура: Реализация в графическом редакторе
1	В менеджере тегов создайте тег типа *Signed 32–Bit Value (32–битная величина со знаком)*. В данном примере используется тег *S32i_varia_but_00*.
2	В кадр поместите *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В данном примере используется объект *I/O Field1*. При настройке *поля ввода/вывода* в *диалоге конфигурирования* укажите тег *S32i_varia_but_00*. Поменяйте значение по умолчанию в поле *Update (Обновление)* на *Upon Change (По изменению)* и установите *Field Type (Тип поля)* в *Output (Вывод)*. 
3	В том же кадре разместите *Windows Object (Объект Windows)* → *Button (Кнопку)*. В данном примере используется объект *Button2*.

4	Для изменения уставки создайте *процедуру Cи*, связанную с событием *Event (Событие)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Эта *процедура Cи* изменит значение тега при любом нажатии мышью на кнопку. Предельное значение задается и проверяется также в *процедуре Cи*.
5	Настройка декрементирования уставки производится тем же способом. В данном примере для этого используется объект *Button1*.

Процедура Cи, связанная с кнопкой Button2

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
DWORD value;

value=GetTagDWord("S32i_varia_but_00"); //get tag value
if (value>1300) (value=1400);           //check limit
else value=value+100;                   //inc value
SetTagDWord("S32i_varia_but_J0",value); //set new value
}
```

Объявляется *переменная Cи*.

Для получения текущего значения тега *S32i_varia_but_00* используется *внутренняя функция GetTagDWord*.

В условном операторе *if* проверяется, превышает ли значение тега 1300. Если это так, то в качестве верхнего предела будет задано число 1400. Если значение тега меньше 1300, то в ветви *else* оно увеличивается на 100.

Внутренняя функция SetTagDWord записывает измененное значение обратно в тег *S32i_varia_but_00*.

Замечание относительно основных применений

В общем случае перед применением описанных процедур необходимо указать требуемые теги и модифицировать инкремент и граничные значения.

2.2.2 Дискретное изменение — изменение значения уставки с использованием глобального сценария (example 02)

Постановка задачи

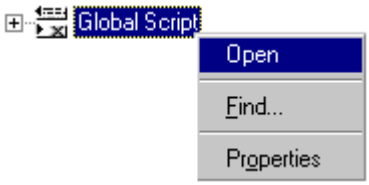
Реализовать дискретное изменение параметра при помощи мыши. Изменение значения уставки на фиксированную величину должно производиться по нажатию кнопки. Изменение значения должно быть ограничено установленными пределам. Изменения должны быть реализованы с использованием *функции проекта*.

Концепция реализации

Для реализации используются две *Windows Object (Объект окна)* → *Buttons (Кнопки)*, с помощью которых организовано событийно–ориентированное изменение уставки. При нажатии на *кнопку* (мышью) значение *внутреннего тега* изменяется на единицу инкремента. Единица инкремента задается предварительно, и не может быть изменена в режиме исполнения. Изменение уставки выполняется в рамках *функции проекта*.

Изменение значения уставки отображается в *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. Выходное значение *поля ввода/вывода* связывается с *внутренним тегом*.

Создание функции проекта

Шаг	Процедура: Создание функции проекта
1	Запустите *редактор глобальных сценариев* в *проводнике WinCC* нажатием на R (правую кнопку мыши) на элементе *Global Script* и выберите пункт *Open (Открыть)* во всплывающем меню. 
2	Создайте новую функцию, выбрав в меню пункт *File (Файл)* → *New Project Function (Новая функция проекта)*.
3	Присвойте функции имя *IncDecValue* и сохраните функцию выбором пункта меню *File (Файл)* → *Save As (Сохранить как)* → *IncDecValue.fct*.
4	Запрограммируйте и откомпилируйте функцию.

Функция проекта IncDecValue

```
void IncDecValue(DWORD *value,DWORD low,DWORD high,DWORD step,DWORD a)

{
DWORD v;
v=*value; //get current value
switch (a){
   case 0: {
          if (v<step) (v=0); //low limit
          else v=v-step;     //decrement
          } //case 0
          break;
   case 1:{
          if (v>(high-step))
             (v=high);       //high limit
          else v=v+step;     //increment
          }//case 1
          break;
   }//switch
*value=v; //return
}
```

Функция имеет заголовок, содержащий название функции проекта *IncDecValue* и передаваемые параметры. Эта *функция проекта* используется для инкрементирования и декрементирования.

Объявляются переменные.

При вызове функции переменные передаются не по значению, а по указателю. Содержимое адреса считывается и помещается *Cи–переменную v*.

При помощи оператора *switch* оценивается состояние переменной направления изменения *a*.

В соответствующей ветви *case* производится проверка граничных значений и определяется минимальное (максимальное) значение при выходе за границы.

Если выхода за граничные значения не было, то текущее значение изменяется.

Текущее значение уставки заносится по адресу переменной, подлежащей обработке.

Реализация в графическом редакторе

Шаг	Процедура: Реализация в графическом редакторе
1	В менеджере тегов создайте тег типа *Signed 32–Bit Value (32–битная величина со знаком)*. В данном примере используется тег *S32i_varia_but_04*.
2	В кадре сконфигурируйте *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В данном примере используется поле ввода/вывода *I/O Field2*. При настройке *поля ввода/вывода* в *диалоге конфигурирования* укажите переменную *S32i_varia_but_04*. Поменяйте значение по умолчанию в поле *Update (Обновление)* на *Upon Change (По изменению)* и установите *Field Type (Тип поля)* в *Output (Вывод)*.
3	В том же кадре настройте *Windows Object (Объект Windows)* → *Button (Кнопка)*. В этом примере используется кнопка *Button7*.

4	Для изменения уставки создайте *процедуру Си*, связанную с событием *Event (Событие)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Эта *процедура Си* вызывает *функцию проекта IncDecValue* и передает ей требуемые параметры. Изменение значения тега происходит при каждом щелчке мыши на *кнопке*. Предельное значение определяется передающимся в *функцию проекта* параметром во время ее вызова. Проверка выполняется в *функции проекта*.
5	Настройка декрементирования уставки происходит точно так же. В примере для этого используется кнопка *Button6*.

Процедура Си, связанная с кнопкой Button7

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
DWORD value;

value=GetTagDWord("S32i_varia_but_04");

//IncDecValue(DWORD *value,DWORD low,DWORD high,DWORD step,DWORD a )
IncDecValue(&value,0,1400,100,1);
SetTagDWord("S32i_varia_but_04",value);
}
```

Для чтения текущего значения *внутреннего тега* используется *внутренняя функция GetTagDWord*.

Вызывается *функция проекта IncDecValue*, ей передаются такие параметры, как указатель на переменную, верхняя и нижняя границы, инкремент, признак.

Для передачи нового значения во *внутренний тег* используется *внутренняя функция SetTagDWord*.

Замечание относительно основных применений

Функция проекта может использоваться сразу после ее создания, и не требовать каких либо доработок в дальнейшем. В *процедуре Си*, использующейся для вызова *функции проекта*, необходимо в соответствии с вашими требованиями сформировать список передаваемых параметров.

2.2.3 Дискретное изменение — кнопка (example 05)

Приемы, описываемые в данной главе, доступны для изучения в проекте *Project_TagHandling* по нажатию (мышью) на изображенные выше *кнопки*. Они приведены в кадре *pictu_3_chapter_01a.pdl*.

Постановка задачи

Реализовать дискретное изменение с использованием мыши.
Устройство (двигатель, задвижка) должно активизироваться по щелчку на кнопке. При отпускании кнопки устройство следует отключить.

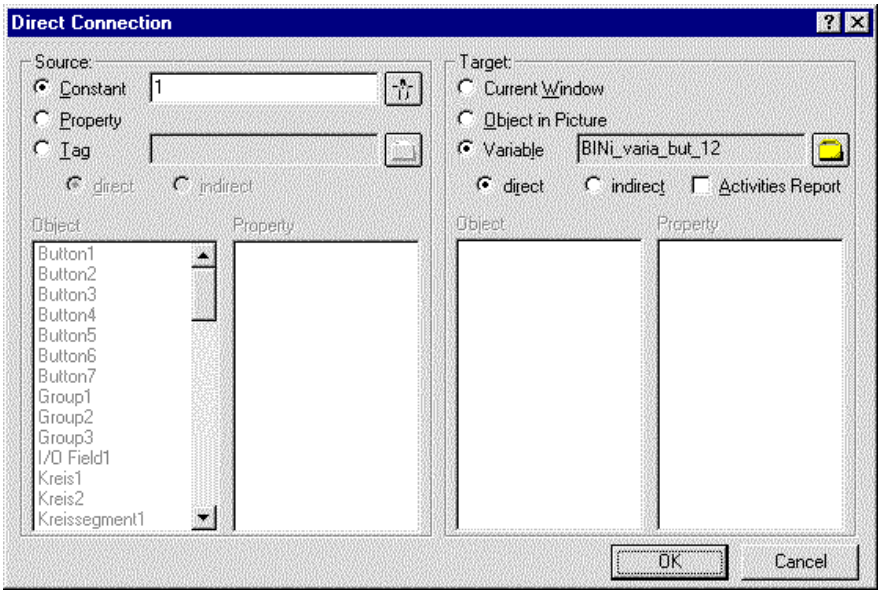
Концепция реализации

Событийно–управляемая кнопка создается на основе элемента *Windows Object (Объект Windows)* → *Button (Кнопка)*. Визуализация производится с использованием *Direct Connection (Прямого соединения)* и *C–Action (Процедуры Cи)*.

Примечание:
Настройка кнопки с использованием *прямого соединения* является наиболее эффективным методом с точки зрения производительности в режиме исполнения.

Реализация в графическом редакторе — прямое соединение

Шаг	**Процедура: Прямое соединение**
1	В менеджере тегов создайте тег типа *Binary Tag (Двоичный тег)*. В данном примере используется тег *BINi_varia_but_12*.
2	В кадр поместите элемент *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется кнопка *Button2*.

Шаг	Процедура: Прямое соединение
3	У кнопки *Button2* настройте *прямое соединение* для события *Event (Событие)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Свяжите элемент *Source Constant (Источник, константа)* → *1* с *Target Tag (Приемник, тег)* → *BINi_varia_but_12*. Подтвердите настройки нажатием на кнопку *OK*. Сконфигурируйте еще одно *прямое соединение* для события *Event (Событие)* → *Mouse (Мышь)* → *Release Left (Отпускание левой кнопки)*, но на этот раз для элемента *Source Constant (Источник, константа)* → *0*. 
4	Анимация управляется тегом *BINi_varia_but_12*.

Ниже дополнительно приводится описание выполнения этого же задания с использованием *процедуры Си*. Однако, описанный выше метод с использованием *прямого соединения* работает лучше и быстрее.

Реализация в графическом редакторе — процедура Си (C–Action)

Шаг	Процедура: Процедура Си
1	В менеджере тегов создайте тег типа *Binary Tag (Двоичный тег)*. В данном примере используется тег *BINi_varia_but_12*.
2	В кадр поместите объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется объект *Button1*.
3	С событием *Event (Событие)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* свяжите *процедуру Си*, которая устанавливает значение тега *BINi_varia_but_12* в *1*. С событием *Event (Событие)* → *Mouse (Мышь)* → *Release left (Отпускание левой кнопки)* свяжите другую *процедуру Си*, которая устанавливает значение тега *BINi_varia_but_12* в *0*.

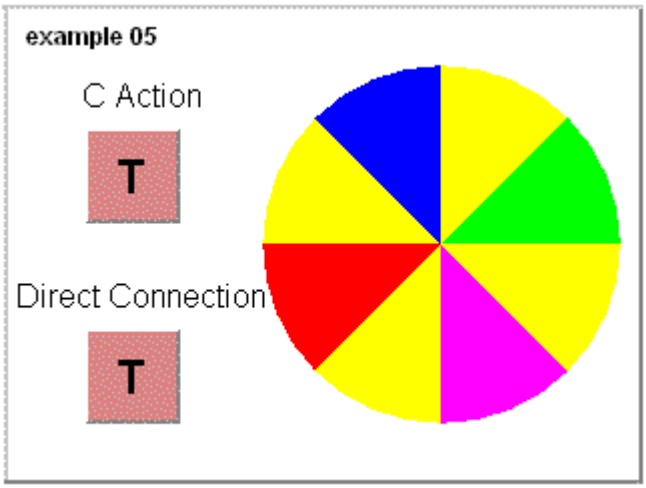
Процедура Си (C–Action), связанная с кнопкой Button1

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetTagWord("BINi_varia_but_12",1); //on
}
```

Для установки тега в *1* используется *внутренняя функция SetTagDWord*.

Анимация на примере

В данном примере, кнопка используется для анимации цветного круга.



Цветной круг состоит из нескольких объектов *Standard Objects (Стандартные объекты)* → *Pie Segments (Сегменты круга)*.

Динамические свойства придаются объектам при помощи *динамического диалога*, сконфигурированного для атрибутов *Properties (Свойства)* → *Geometry (Геометрия)* → *Start Angle (Начальный угол)* и *Properties (Свойства)* → *Geometry (Геометрия)* → *End Angle (Конечный угол)*. Для решения задачи нам необходимо действие, модифицирующее значение угла поворота в определенные моменты времени. Для *Pie Segment4* мы описываем изменение значения с использованием *процедуры Си*, связанной с атрибутом *Property (Свойство)* → *Colors (Цвета)* → *Line Color (Цвет линии)*. Период срабатывания процедуры устанавливается равным *250 ms*. В этом случае мы не делаем *цвет линии* динамически изменяемым. Причиной использования *процедуры Си* для этого свойства является необходимость наличия триггера для изменения значения. Вместо этого свойства мы также могли бы использовать другие атрибуты объекта.

Текущий угол поворота изменяется во *внутреннем теге S32i_vara_but_11*.

Процедура Си (C–Action) для анимации

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN

{
Static DWORD i = 0;

//if button pressed
if (GetTagBit("BINi_varia_but_12")) {
   i=i+10;              //increment of rotation
   if (i==360) (i=0);  //high limit
   SetTagDWord("S32i_varia_but_11",i);
   }//if
return(0x0); //black
}
```

Переменная Си i объявляется как *static DWORD*, так как ее значение должно оставаться неизменным, пока открыта экранная форма.

Если *кнопка* нажата, круг вращается с шагом в 10 градусов, т.е. значение тега инкрементируется на 10.

Тег *i* инициализируется после каждого полного оборота (360°).

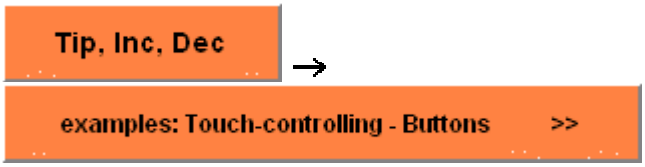
Новое значение угла поворота передается во *внутренний тег*.

Заданное значение цвета фона возвращается по команде *return*. Его не следует изменять.

Замечание относительно основных применений

В общем случае перед использованием описанной *кнопки* с *прямым соединением* необходимо указать нужный тег.

2.2.4 Дискретное изменение — двухполюсный переключатель (example 06)



Пример, имеющий отношение к данной главе, приведен в проекте *Project_TagHandling* и доступен по нажатию (мышью) на *кнопке*, показанной выше. Он приведен в кадре *pictu_3_chapter_01a.pdl*.

Постановка задачи

Дискретное изменение необходимо реализовать с использованием мыши.
Функцию двухполюсного переключателя следует реализовать с использованием кнопки.
Нажатие на кнопку должно включать устройство (двигатель, задвижку) и после отпускания кнопки оно должно оставаться включенным. Выключение устройства осуществляется повторным нажатием на кнопку.

Концепция реализации

Событийно–управляемый двухполюсный переключатель реализован на основе объекта *Windows Object (Объект Windows)* → *Button (Кнопка)*.

Примечание:
Реализация ключа переключения посредством *прямого соединения* является более эффективной, но для этого требуются две кнопки.

Реализация в графическом редакторе — прямое соединение

Шаг	Процедура: Прямое соединение
1	В менеджере тегов создайте тег типа *Binary Tag (Двоичный тег)*. В данном примере используется тег *BINi_varia_but_16*.
2	В кадр поместите два объекта *Windows Objects (Объект Windows)* → *Buttons (Кнопка)*. В этом примере для включения используется кнопка *Button4*, а для выключения — *Button5*.
3	Для *Button4* настройте *прямое соединение* с событием *Event (Событие)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Свяжите *Source Constant (Источник, константа)* → *1* для *Target Tag (Приемник, тег)* → *BINi_varia_but_16*. Подтвердите установки нажатием на кнопку *OK*. Для *Button5* настройте *прямое соединение* таким же образом, только с *Source Constant (Источник, константа)* → *0*.
4	*Прямое соединение* с событием *Event (Событие)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* осуществляет лишь синхронизацию маркировки *Button3* и не требуется для реализации функциональных возможностей переключателя.

Реализация в графическом редакторе — процедура Си (C–Action)

Шаг	Процедура: Процедура Си
1	В менеджере тегов создайте тег типа *Binary Tag (Двоичный тег)*. В данном примере используется тег *BINi_varia_but_16*.
2	В кадр поместите объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется кнопка *Button3*.
3	С событием *Event (Событие)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*, свяжите *процедуру Си*, которая инвертирует тег *BINi_varia_but_16*.

Процедура Си (C–Action) для двухполюсного переключателя

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
BOOL state;

//flip tag
state = !GetTagBit("BINi_varia_but_16");
SetTagBit("BINi_varia_but_16",(SHORT)state);
}
```

Объявляется переменная *state*.

С помощью *внутренней функции GetTagBit* значение *внутреннего тега* считывается, инвертируется и возвращается посредством функции *SetTagBit*.

Замечание относительно основных применений

В общем случае перед использованием описанной *кнопки* с *процедурой Си* необходимо указать нужную переменную. Инверсия *внутреннего тега* может быть также выполнена без использования *переменной Си*, как показано ниже:

```
SetTagDWord("BINi_varia_but_16",
(SHORT)!GetTagBit("BINi_varia_but_16"));
```

2.2.5 Инкрементирование и декрементирование (example 01)

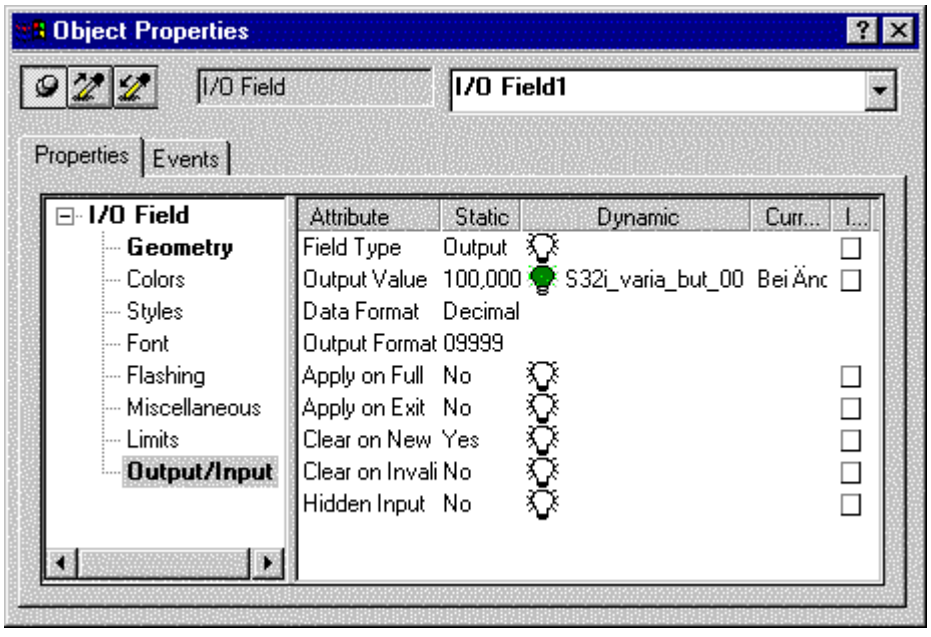
Пример, имеющий отношение к данной главе, приведен в проекте *Project_TagHandling* и доступен по нажатию (мышью) на *кнопке*, показанной выше. Он приведен в кадре *pictu_3_chapter_01.pdl*.

Постановка задачи

Организовать изменение значения тега. Изменение значения должно производиться в определенных пределах и выполняться с использованием мыши. Значение тега должно изменяться по нажатию кнопки. Значение следует изменять только при нажатии кнопки. Установленное значение должно сохраняться при отпускании кнопки.

Концепция реализации

Для создания событийно–управляемой кнопки выберите объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. При щелчке (мышью) на кнопке, значение *внутреннего тега* увеличивается на заданную единицу инкремента, а при щелчке на ней R (правой кнопкой мыши) его значение уменьшается на единицу инкремента. Значение изменяется, пока кнопка находится в нажатом состоянии. Инкремент задается предварительно и не может быть изменен в режиме исполнения.
Для отображения изменяющегося значения используется *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. Выходное значение в *поле ввода/вывода* связывается с *внутренним тегом*.

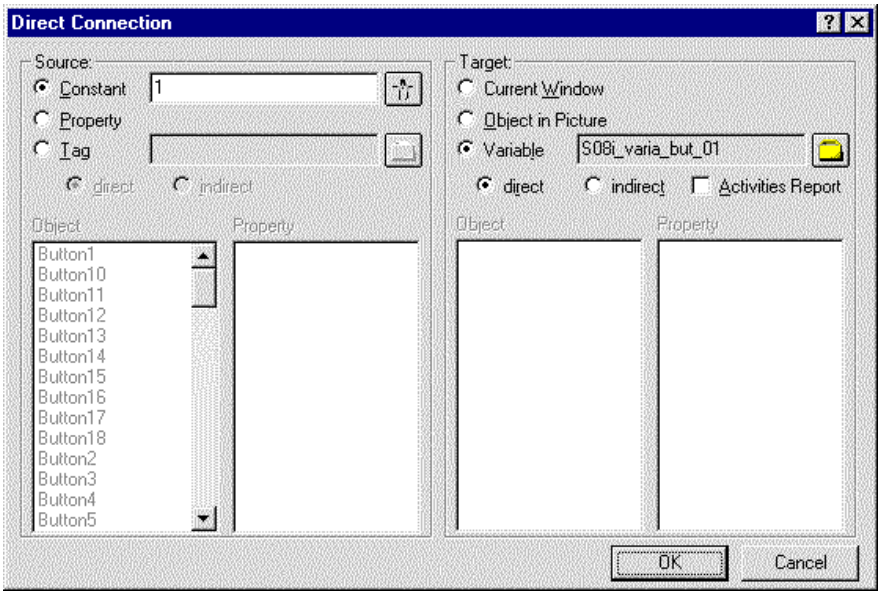


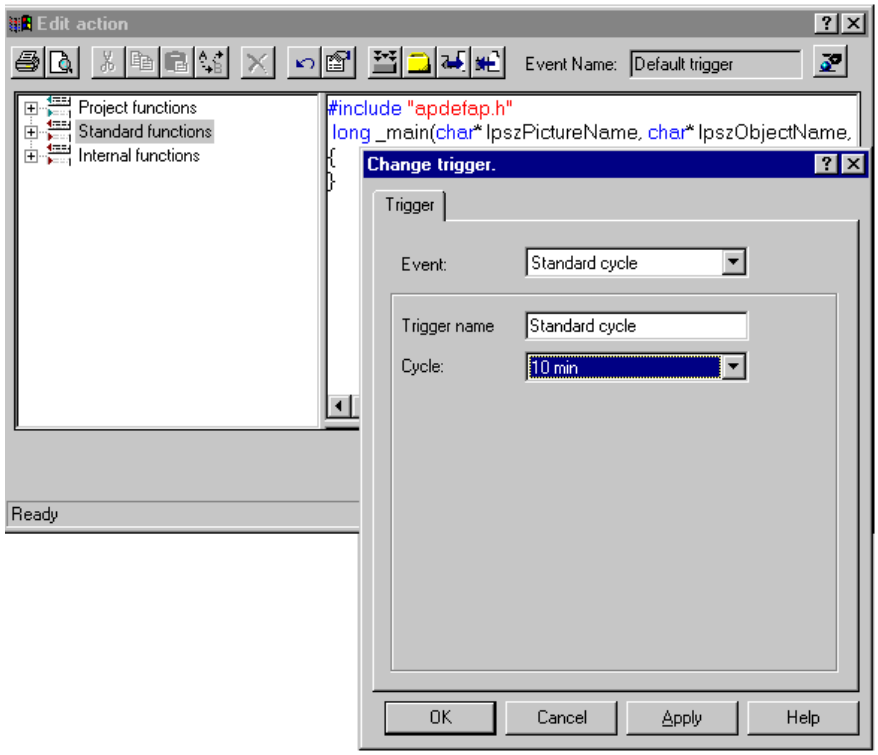
Изменение значения

Для выполнения задачи необходимо, чтобы определенное действие, осуществляющее модификацию значения *внутреннего тега*, исполнялось в заданные моменты времени. Изменение значения выполняется в *процедуре Cu*, связанной с атрибутом *Property (Свойство)* → *Geometry (Геометрия)* → *Position X (Координата X) поля ввода/вывода*. Время срабатывания устанавливается равным *250 ms*. Мы **не** делаем положение *поля ввода/вывода* динамическим. *Процедура Cu* связывается с этим свойством по той причине, что мы хотим реализовать изменение значения непосредственно в рамках самого объекта.
В данном демонстрационном проекте, эта проблема также решается с использованием *глобальной процедуры (Global Action)*.

Реализация в проекте WinCC

Шаг	Процедура: Инкрементирование, декрементирование
1	В менеджере тегов создайте теги. В данном примере используются теги *S32i_varia_but_00* и *S08i_varia_but_01*.
2	В кадр поместите *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В данном примере используется объект *I/O Field1*. Во время создания *I/O Field* в *диалоге конфигурирования* укажите тег *S32i_varia_but_00*. Поменяйте значение по умолчанию в поле *Update (Обновление)* на *Upon Change (По изменению)* и установите *Field Type (Тип поля)* в *Output (Вывод)*.
3	В тот же кадр поместите объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В этом примере используется кнопка *Button3*.

4	Для изменения уставки по щелчку мыши с этой *кнопкой* устанавливается несколько *прямых соединений*. Эти *прямые соединения* изменяют значение тега *S08i_varia_but_01* каждый раз, когда происходит нажатие на *кнопку* (левой) или R (правой кнопкой мыши). С событием *Event (Событие)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* свяжите установку инкрементирования в ON (ВКЛ) (установка тега в *1*). С событием *Event (Событие)* → *Mouse (Мышь)* → *Release left (Отпускание левой кнопки)* свяжите установку инкрементирования в OFF (ВЫКЛ) (установка тега в *0*). С событием *Event (Событие)* → *Mouse (Мышь)* → *Press right (Нажатие правой кнопки)* свяжите установку декрементирования в ON (ВКЛ) (установка тега в *2*) и с событием *Event (Событие)* → *Mouse (Мышь)* → *Release right (Отпускание правой кнопки)* свяжите установку декрементирования в OFF (ВЫКЛ) (установка тега в *0*). 
5	Изменение значения тега *S32i_varia_but_00* выполняется в *процедуре Cu*, связанной с атрибутом *Property (Свойство)* → *Geometry (Геометрия)* → *Position X (Координата X)* объекта *I/O Field1*.

6	Периодичность вызова *процедуры Си* изменена на *250 ms*. 

Процедура Си, связанная с изменением значения I/O Field

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN

{
DWORD value;
SHORT count;

count = GetTagWord("S08i_varia_but_01");  //inc or dec
if ((count==1) || (count==2)){
   //current value
   value = GetTagDWord("S32i_varia_but_00");

   if (count==1){ //inc
       value++;
       if (value>1400) (value=1400);       //high limit
       SetTagDWord("S32i_varia_but_00",value);
       }//inc
   if (count==2){ //dec
       value--;
       if (value<0) (value=0);             //low limit
       SetTagDWord("S32i_varia_but_00",value);
       }//dec
   }//if count
return(81); //x-pos
}
```

Объявляются *переменные Си value* и *count*.

Выясняется, нажата ли *кнопка*. Если *кнопка* не нажата, *процедура Си* завершается (во избежание ненужной загрузки системы).

Если *кнопка* нажата, процедура определяет, должно значение инкрементироваться или декрементироваться. В соответствии с полученным результатом производится изменение значения тега.

После изменения величины производится проверка граничного значения.

По команде *return* возвращается значение, заданное для координаты X. Оно не подлежит изменению.

Замечание относительно основных применений

В общем случае перед использованием описанной *кнопки* с *прямым соединением* необходимо указать нужную переменную и модифицировать *процедуру Си* для *поля ввода/вывода*. В *процедуре Си* необходимо произвести согласование предельных значений и переменных.

2.2.6 Инкрементирование и декрементирование с помощью глобальных процедур (example 02)

Постановка задачи

Организовать изменение значения тега. Изменения должны производиться в заданных пределах и выполняться с помощью мыши.
Значение тега следует изменять при нажатии кнопки. Значение должно изменяться только по нажатию кнопки. Установленное значение должно быть сохранено при отпускании кнопки.

Концепция реализации

Для создания событийно–управляемой кнопки используйте объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. Реализация основана на использовании *глобальной процедуры*.

При нажатии на кнопку (мышью), значение *внутреннего тега* увеличивается на заданную единицу инкремента, а при нажатии R (правой кнопкой мыши), значение уменьшается на ту же величину. Значение изменяется, пока кнопка находится в нажатом состоянии. Инкремент задается предварительно и не может быть изменен в режиме исполнения.
Для отображения изменяющегося значения используется *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. Выходное значение в *поле ввода/вывода* связывается с *внутренним тегом*.

Изменение значения

Для выполнения задачи необходимо, чтобы определенное действие, осуществляющее модификацию значения *внутреннего тега*, исполнялось в заданные моменты времени. Для изменения значения мы будем использовать *глобальную процедуру*.
Процедура активизируется при запуске режима исполнения WinCC и далее исполняется с заданной цикличностью. Действие программируется таким образом, что основная часть программы выполняется фактически только при нажатой кнопке.
Необычным моментом является использование внешних переменных Си. Внешние Си–переменные действительны в течение всего времени исполнения проекта WinCC, но они должны быть объявлены вне заголовка функции. В WinCC это возможно только в функции проекта, для объявления таких тегов создана отдельная функция проекта. Эта функция проекта должна выполняться один раз при запуске проекта.

Создание функции проекта

Шаг	Процедура: Создание функции проекта
1	В *проводнике WinCC* запустите редактор глобальных процедур *Global Script*.
2	Создайте новую функцию, выбрав в меню пункт *File (Файл)* → *New Project Function (Новая функция проекта)*.
3	Задайте функции имя *InitAction* и сохраните ее, выбрав в меню пункт *File (Файл)* → *Save As (Сохранить как)* → *InitAction.fct*.
4	Запрограммируйте и откомпилируйте функцию.

Функция проекта InitAction

```
//declaration for counter.pas
extern char tagname[30] = " ";
extern SHORT count = 0;
extern DWORD low = 0;
extern DWORD high = 0;
extern DWORD step = 0;

void InitAction()
{
//function is used to generate external tags
}
```

Объявляются внешние *переменные Си*.

Данная функция должна быть выполнена только один раз при запуске проекта. Вызов функции рекомендуется производить в стартовом кадре, по событию *Event (Событие)* → *Miscellaneous (Разные)* → *Open Picture (Открытие кадра)*.

Создание глобальной процедуры

Шаг	Процедура: Создание глобальной процедуры
1	В *проводнике WinCC* запустите редактор глобальных процедур *Global Script*.
2	Создайте новую процедуру, выбрав в меню пункт *File (Файл)* → *New Action (Новая процедура)*.
3	Сохраните файл, выбрав пункт меню *File (Файл)* → *Save As (Сохранить как)* → *counter.pas*.
4	Запрограммируйте и откомпилируйте *процедуру*.
5	Задайте триггер. Это делается при помощи *кнопки*  на панели инструментов. В диалоговом окне *Description (Описание)*, выберите закладку *Trigger (Триггер)*. Добавьте *Timer (Таймер)* → *Standard Cycle (Стандартный цикл)* → *250 ms*.

Глобальная процедура counter.pas

```
#include "apdefap.h"

int gscAction( void )
{
extern char tagname[30];
extern SHORT count;
extern DWORD low;
extern DWORD high;
extern DWORD step;

DWORD value;

if ((count==1)||(count==2)) {
        //get current value
        value = GetTagDWord(tagname);
        if (count==1){                          //inc
                value = value+step;
                if (value>high) (value=high); //high limit
                }//if
        if (count==2){                           //dec
                value = value-step;
                if (value<low) (value=low);  //low limit
                }//if
        SetTagDWord(tagname,value);
        }//if
return(0);
}
```

Объявляется внешняя *переменная Си*.

Выясняется, нажата ли *кнопка*. Если *кнопка* не нажата, *процедура Си* завершается (во избежание ненужной загрузки системы).

Если *кнопка* нажата, процедура определяет, должно значение инкрементироваться или декрементироваться. В соответствии с полученным результатом производится изменение значения тега.

После изменения значения производится проверка граничного значения.

Для присвоения обрабатываемой переменной нового значения используется *внутренняя функция SetTagDWord*.

Реализация в графическом редакторе

Шаг	Процедура: Реализация в графическом редакторе
1	В менеджере тегов создайте тег. В данном примере используется тег *S32i_varia_but_04*.
2	В кадре разместите *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В данном примере используется объект *I/O Field2*. При настройке *поля ввода/вывода* в *диалоге конфигурирования* укажите тег *S32i_varia_but_04*. Поменяйте значение по умолчанию в поле *Update (Обновление)* на *Upon Change (По умолчанию)* и установите *Field Type (Тип поля)* в *Output (Вывод)*.
3	В тот же кадр поместите *Windows Object (Объект Windows)* → *Button (Кнопку)*. В данном примере используется кнопка *Button8*.

4	Для изменения уставки по щелчку мыши, создайте для этой *кнопки* несколько *процедур Си*. С событием *Event (Событие)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* свяжите установку инкрементирования в ON (ВКЛ). С событием *Event (Событие)* → *Mouse (Мышь)* → *Release left (Отпускание левой кнопки)* свяжите установку инкрементирования в OFF (ВЫКЛ). С событием *Event (Событие)* → *Mouse (Мышь)* → *Press right (Нажатие правой кнопки)* свяжите установку декрементирования в ON (ВКЛ), и с событием *Event (Событие)* → *Mouse (Мышь)* → *Release right (Отпускание правой кнопки)* свяжите установку декрементирования в OFF (ВЫКЛ). Эти *процедуры Си* передают необходимые параметры *глобальной процедуре counter.pas*. Это происходит при каждом нажатии (левой) или R (правой кнопки мыши).
5	Изменение значения тега *S32i_varia_but_04* выполняется в *глобальной процедуре counter.pas*.

Процедура Си, связанная с Button8 для включения инкрементирования

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
//inc on
extern char tagname[30];
extern SHORT count;
extern DWORD low;
extern DWORD high;
extern DWORD step;

strcpy(tagname,"S32i_varia_but_04");
count = 1;
low = 0;
high =1400;
step = 1;
}
```

Процедура Си, связанная с Button8 для отключения инкрементирования

```
#include "apdefap.h"
void OnLButtonUp(char* lpszPictureName, char* lpszObjectName, char* lpszPrc

{
//inc off
extern SHORT count;
count=0;
}
```

В *процедуре Си* объявляются внешние *переменные Си*. Эти переменные созданы в функции проекта *InitAction*.

В переменные заносятся необходимые значения. Это сопоставимо с передачей параметров *функции проекта*. Содержимое переменной *count* представляет результаты работы кода *глобальной процедуры*.

При выключении инкрементирования задавать значения всех тегов не обязательно.

Замечание относительно основных применений

В общем случае для применения описанного подхода необходимо проделать следующее:

В *процедуре Cи* изменить название тега и установить требуемые граничные значения и инкремент.

Если данная кнопка перенесена в другой проект, *функция проекта InitAction* и *глобальная процедура counter.pas* также должны быть скопированы вместе с ней.

2.2.7 Остальные примеры главы

Example 03

Функционально этот пример похож на *example 01*. Основное отличие состоит в том, что инкремент может быть изменен в режиме исполнения.
Другое отличие — динамическое изменение инкремента во время его установки. Если инкремент > 20, значение изменяется с шагом в 10 единиц; а если < 20, то значение изменяется по 1 единице.

Example 04

Функционально этот пример является комбинацией *example 01* и *example 02*. Значение изменяется с помощью *глобальной процедуры counter.pas*.

Example 07

Функционально этот пример подобен *example 05*. Отличие заключается в режиме анимации.

2.3 Изменение значений тегов с помощью элементов управления

Приемы, описываемые в данной главе, доступны для изучения в проекте *Project_TagHandling* по нажатию (мышью) на изображенной выше *кнопке*. Они приведены в кадре *varia_3_chapter_02.pdl*.

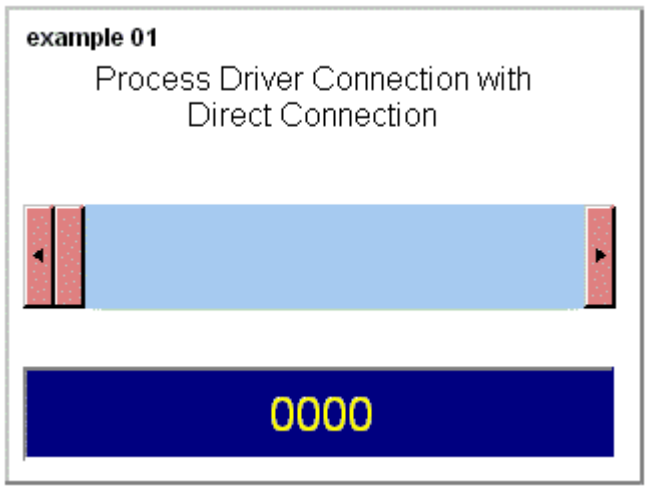
2.3.1 Ввод при помощи бегунка с прямым соединением (example 01)

Постановка задачи

Реализовать изменение уставки с использованием бегунка.
Это изменение должно производиться в заданных пределах.

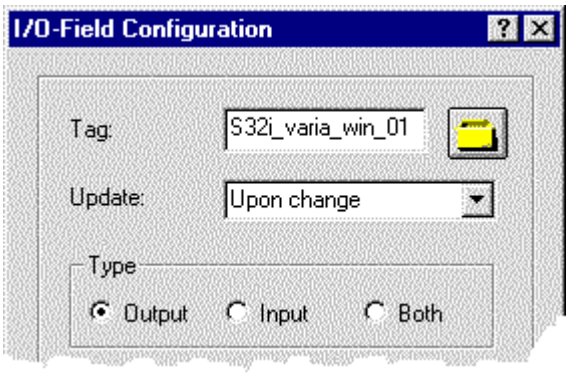
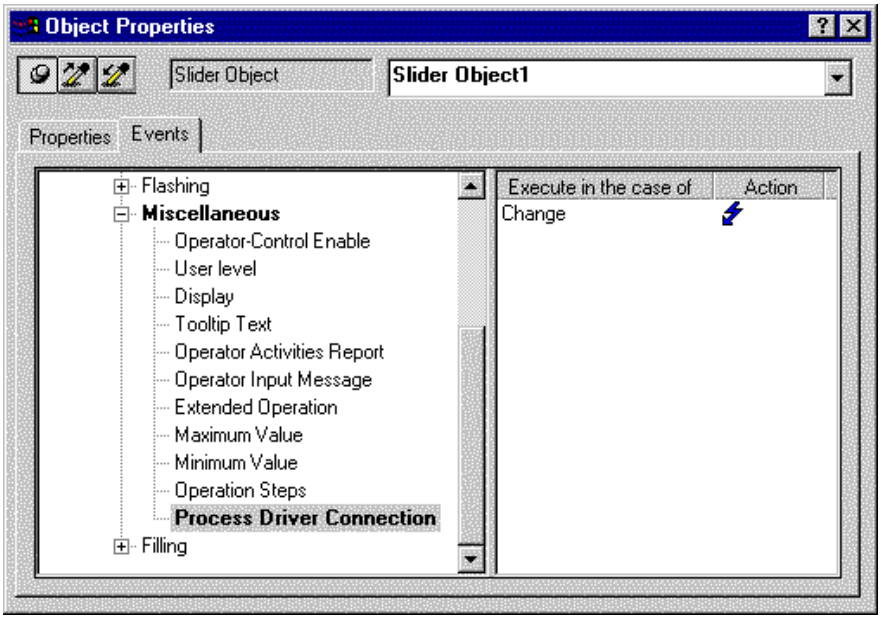
Концепция реализации

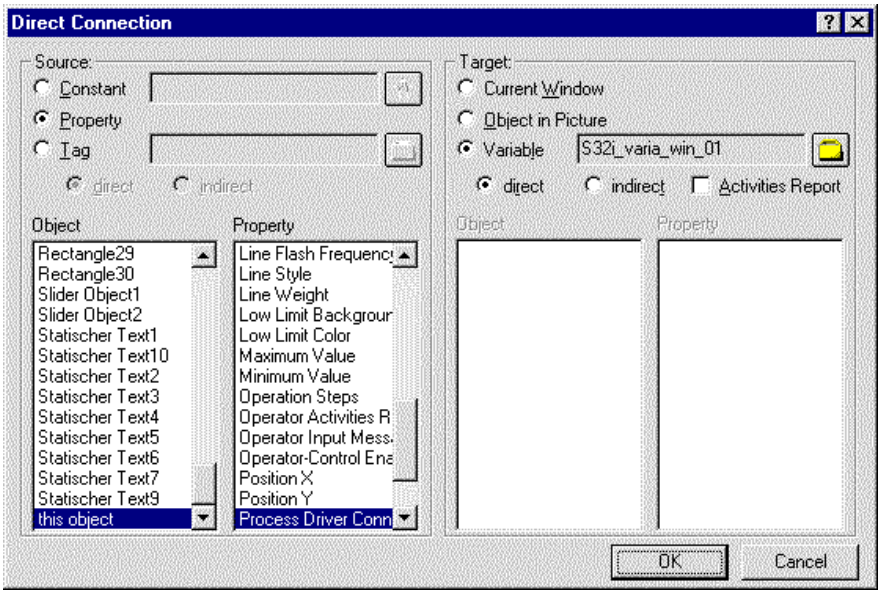
Для реализации изменения уставки мы будем использовать объект *Windows Object(Объект Windows)* → *Slider Object (Бегунок)*. Значение *внутреннего тега* изменяется при перемещении бегунка посредством использования *прямого соединения*.
Изменение уставки отображается в *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*.



Реализация в графическом редакторе

Шаг	Процедура: Реализация в графическом редакторе
1	В менеджере тегов создайте тег. В данном примере используется тег *S32i_varia_win_01*.

Шаг	Процедура: Реализация в графическом редакторе
2	В кадр поместите *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В данном примере используется объект *I/O Field1*. При создании *поля ввода/вывод* в *диалоге конфигурирования* укажите тег *S32i_varia_win_01*. Поменяйте значение по умолчанию в поле *Update (Обновление)* на *Upon Change (По изменению)* и установите *Field Type (Тип поля)* в *Output (Вывод)*. 
3	В том же кадре поместите *Windows Object (Объект Windows)* → *Slider Object (Бегунок)*. В данном примере используется объект *Slider Object1*. Для события *Event (Событие)* → *Miscellaneous (Разные)* → *Process Driver Connection (Соединение с драйвером процесса)* создайте *прямое соединение*. 

Шаг	Процедура: Реализация в графическом редакторе
4	В диалоге *Direct Connection (Прямое соединение)* установите связь между *Source this object (Источник, этот объект)* → *Process Driver Connection (Соединение с драйвером процесса)* и *Target Variable (Приемник, переменная)* → *S32_varia_win_01*. Подтвердите изменения нажатием на кнопку *OK*. 

Замечание относительно основных применений

В общем случае для применения описанного подхода необходимо проделать следующее:

Изменить тег для *прямого соединения*.

Изменить диапазон изменения значения *бегунка* в пунктах *Properties (Свойства)* → *Miscellaneous (Разные)* → *Maximum Value (Максимальное значение)* и *Minimum Value (Минимальное значение)*. Это также можно сделать в *диалоге конфигурирования* бегунка.

2.3.2 Ввод при помощи бегунка и соединения с тегом (example 03)

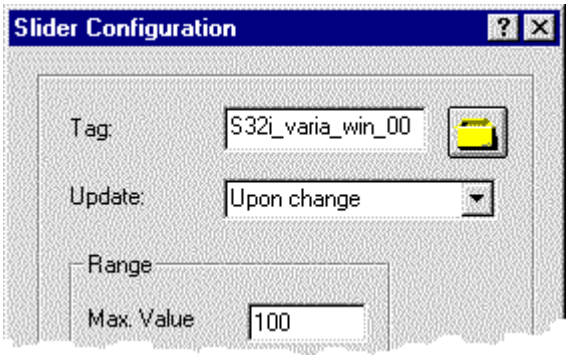
Постановка задачи

Организовать изменение уставки с использованием бегунка.
Это изменение должно производиться в заданных пределах.

Концепция реализации

Для реализации изменения уставки мы будем использовать *Windows Object (Объект Windows)* → *Slider Object (Бегунок)*. Значение *внутреннего тега* изменяется при перемещении бегунка посредством использования *соединения с тегом*. Значение тега записывается только при отпускании бегунка.
Изменение уставки отображается в *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*.

Реализация в графическом редакторе

Шаг	Процедура: Изменение уставки с использованием бегунка — соединение с тегом
1	В менеджере тегов создайте тег. В данном примере используется тег *S32i_varia_win_00*.
2	В кадр поместите *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В данном примере используется объект *I/O Field3*. При создании *поля ввода/вывод* в *диалоге конфигурирования* укажите тег *S32i_varia_win_00*. Поменяйте значение по умолчанию в поле *Update (Обновление)* на *Upon Change (По изменению)* и установите *Field Type (Тип поля)* в *Output (Вывод)*.
3	В том же кадре разместите *Windows Object (Объект Windows)* → *Slider Object (Бегунок)*. В данном примере используется *Slider Object2*. При создании *бегунка* в *диалоге конфигурирования* укажите тег *S32i_varia_win_00*. Поменяйте значение по умолчанию в поле *Update (Обновление)* на *Upon Change (По изменению)*. 

Замечание относительно основных применений

В общем случае для применения описанного подхода необходимо проделать следующее:

Изменить тег для *соединения*.

Изменить диапазон изменения значения *Slider Object (Бегунка)* в пункте *Properties (Свойства)* → *Miscellaneous (Разные)* → *Maximum Value (Максимальное значение)* и *Minimum Value (Минимальное значение)*. Это также можно сделать в *диалоге конфигурирования* бегунка.

2.3.3 Ввод при помощи группы выбора (radio–button) (example 02)

Постановка задачи

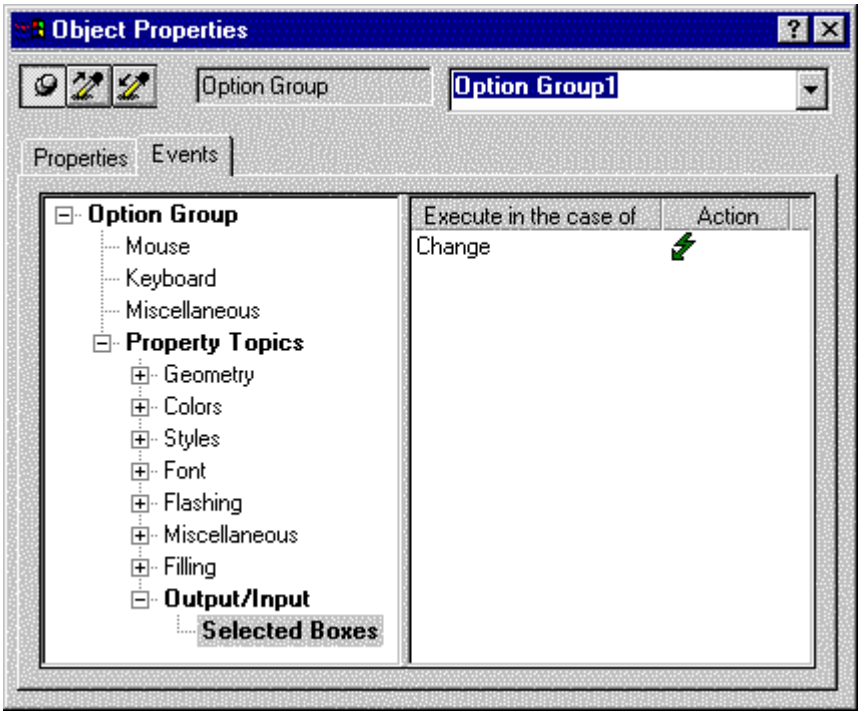
Организовать изменение уставки методом выбора элемента из списка фиксированных значений.

Концепция реализации

Для реализации изменения уставки мы будем использовать объект *Windows Object (Объект Windows)* → *Option Group (Группа выбора)*. Значение *внутреннего тега* изменяется при выборе одной из заданных уставок (мышью). Список установочных значений задан заранее и не может быть изменен в режиме исполнения.
Изменение уставки отображается в *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. Выходное значение *поля ввода/вывода* связывается с *внутренним тегом*. Изменение уставки реализовано в рамках *процедуры Си*.

Реализация в проекте WinCC

Шаг	Процедура: Изменение уставки с помощью группы выбора
1	В менеджере тегов создайте тег. В данном примере используется тег *S32i_varia_win_02*.
2	В кадр поместите *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В данном примере используется объект *I/O Field2*. При создании *поля ввода/вывода* в *диалоге конфигурирования* укажите тег *S32i_varia_win_02*. Поменяйте значение по умолчанию в поле *Update (Обновление)* на *Upon Change (По изменению)* и установите *Field Type (Тип поля)* в *Output (Вывод)*.
3	В том же кадре поместите *Windows Object (Объект Windows)* → *Option Group (Группу выбора)*. В данном примере используется *Option Group1*. В пункте *Property (Свойство)* → *Geometry (Геометрия)* → *Number of Boxes (Количество ячеек)* поменяйте значение по умолчанию с 3 на *4*.
4	Выберите индекс 1 в *Properties (Свойства)* → *Font (Шрифт)* → *Index (Индекс)* → *1*. Введите подходящий текст для выбранного в *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текста)* → *0* элемента. Таким же образом настройте значения для остальных элементов.

Шаг	Процедура: Изменение уставки с помощью группы выбора
5	Для события *Events (События)* → *Property Topics (Разделы свойств)* → *Output/Input (Вывод/Ввод)* → *Selected Boxes (Выбранные пункты)*, создайте *процедуру Си*, которая будет устанавливать значение тега *S32i_varia_win_02* в зависимости от выбранного в группе пункта. 

Процедура Си для группы выбора

```
Void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
//set tag according to selected box
switch(value){
  case 1 : SetTagDWord("S32i_varia_win_02",0);
  break;
  case 2 : SetTagDWord("S32i_varia_win_02",50);
  break;
  case 4 : SetTagDWord("S32i_varia_win_02",100);
  break;
  case 8 : SetTagDWord("S32i_varia_win_02",150);
  break;
  }//switch
}
```

Значения присваиваются тегу *S32i_varia_win_02*, в соответствии с состоянием входов. Это состояние хранится в предварительно описанной переменной *value*.

Замечание относительно основных применений

В общем случае для применения *группы выбора* необходимо проделать следующее:

Указать требуемый тег в *процедуре Си* для события *Events (События)* → *Property Topics (Разделы свойств)* → *Output/Input (Вывод/ввод)* → *Selected Boxes (Выбранные пункты)*.

2.3.4 Ввод при помощи флажков (checkbox) (example 04)

Постановка задачи

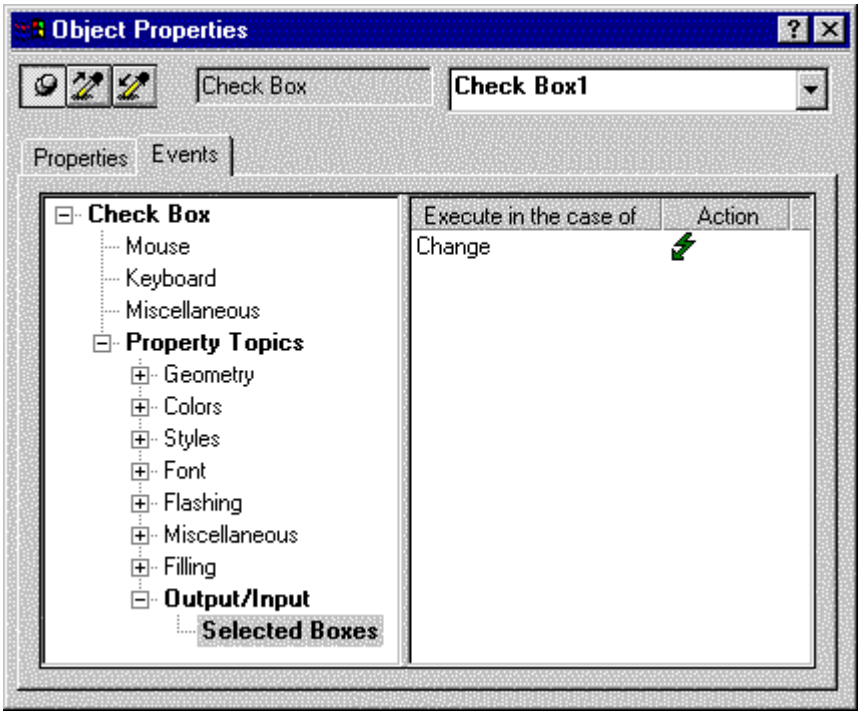
При помощи набора флажков типа checkbox требуется динамически отображать и скрывать различные объекты.

Концепция реализации

Для реализации используется объект *Windows Object (Объект Windows)* → *Check–Box (Набор флажков)*, который устанавливает отдельные биты тега. Объекты *Standard Objects (Стандартные объекты)* → *Polygons (Многоугольники)* отображаются или скрываются в зависимости от значений этих битов. Для отображения выходного двоичного значения набора флажков используется *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*.

Реализация в проекте WinCC

Шаг	Процедура: Ввод с помощью набора флажков
1	В менеджере тегов создайте тег типа *Signed 32–Bit Value (32–битная величина со знаком)*. В данном примере используется тег *S32i_varia_win_03*.
2	В кадр поместите несколько объектов типа *Standard Objects (Стандартные объекты)* → *Polygons (Многоугольники)*. В данном примере используются объекты *Polygon1* – *Polygon7*. Эти объекты могут быть скрыты или отображены в зависимости от состояния *Check–Box (Набора флажков)*.
3	В том же кадре разместите *Windows Object (Объект Windows)* → *Check–Box (Набор флажков)*. В этом примере используется *Check–Box1*. В атрибуте *Property (Свойства)* → *Geometry (Геометрия)* → *Number of Boxes (Количество флажков)*, измените значение с 3 на *7*.
4	Выберите индекс 1 в пункте *Properties (Свойства)* → *Font (Шрифт)* → *Index (Индекс)* → *1*. Введите соответствующий текст для выбранного индекса в поле *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)*. Этот текст является названием объекта, которым вы хотите управлять посредством переключения флажка. Таким же образом настройте значения для других индексов.

Шаг	Процедура: Ввод с помощью кнопки с независимой фиксацией
5	Для события *Events (События)* → *Property Topics (Разделы свойств)* → *Output/Input (Вывод/ввод)* → *Selected Boxes(Выбранные флажки)*, создайте *процедуру Cu*, которая свяжет двоичный статус *Check–Box1* с тегом *S32i_varia_win_03*, и будет управлять отображением объектов типа *Polygon (Многоугольник)*. 
6	Поместите в кадр *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В данном примере используется объект *I/O Field4*. В *диалоге конфигурирования* укажите тег *S32i_varia_win_03*. Поменяйте значение поля *Update (Обновление)* на *Upon Change (По изменению)*. В *Properties (Свойства)* → *Output/Input (Вывод/ввод)*, поменяйте *Data Format (Формат данных)* на *Binary (Двоичный)* и *Output Format (Формат вывода)* на *01111111*.

Процедура Си для набора флажков

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l

{
SetTagDWord("S32i_varia_win_03",value);
//first box selected
if (value&1) SetVisible(lpszPictureName,"Polygon1",1);
else SetVisible(lpszPictureName,"Polygon1",0);
//second box selected
if (value&2) SetVisible(lpszPictureName,"Polygon2",1);
else SetVisible(lpszPictureName,"Polygon2",0);
//third box seleted
if (value&4) SetVisible(lpszPictureName,"Polygon3",1);
else SetVisible(lpszPictureName,"Polygon3",0);
//fourth box selected
if (value&8) SetVisible(lpszPictureName,"Polygon4",1);
else SetVisible(lpszPictureName,"Polygon4",0);
//fifth box selected
if (value&16) SetVisible(lpszPictureName,"Polygon5",1);
else SetVisible(lpszPictureName,"Polygon5",0);
//sixth box selected
if (value&32) SetVisible(lpszPictureName,"Polygon6",1);
else SetVisible(lpszPictureName,"Polygon6",0);
//seventh box selected
if (value&64) SetVisible(lpszPictureName,"Polygon7",1);
else SetVisible(lpszPictureName,"Polygon7",0);
}
```

В тег *S32i_varia_win_03* заносится обновленная информация о состоянии объекта *Check–Box (Набор флажков)*.

В соответствии с их состоянием производится управление видимостью объектов. Состояние записывается в предварительно созданный тег *value*. Для считывания отдельного бита вы должны применить процедуру маскирования.

Примечание:

Похожий пример демонстрируется в проекте *Project_CreatePicture*, в главе *Adding Dynamics (Добавление динамических свойств)*, *example4 (пример4)*. В этом случае, однако, управление видимостью производится для каждого объекта посредством *Dynamic Dialog (Динамического диалога)*.

Замечание относительно основных применений

В общем случае для применения *группы флажков* необходимо проделать следующее:

Указать требуемые названия объектов и имя тега в *процедуре Си* для события *Event (Событие)* → *Property Topics (Разделы свойств)* → *Output/Input (Вывод/ввод)* → *Selected Boxes (Выбранные пункты)*.

2.4 Обработка битов в словах

Приемы, описываемые в данной главе, доступны для изучения в проекте *Project_TagHandling* по нажатию (мышью) на изображенной выше *кнопке*. Они приведены в кадрах *varia_3_chapter_03.pdl* и *varia_3_chapter_03a.pdl*.

Определение

Термин **обработка битов** относится к изменению состояния отдельных битов слова.

2.4.1 Установка бита при помощи флажков и прямого соединения (example 06)

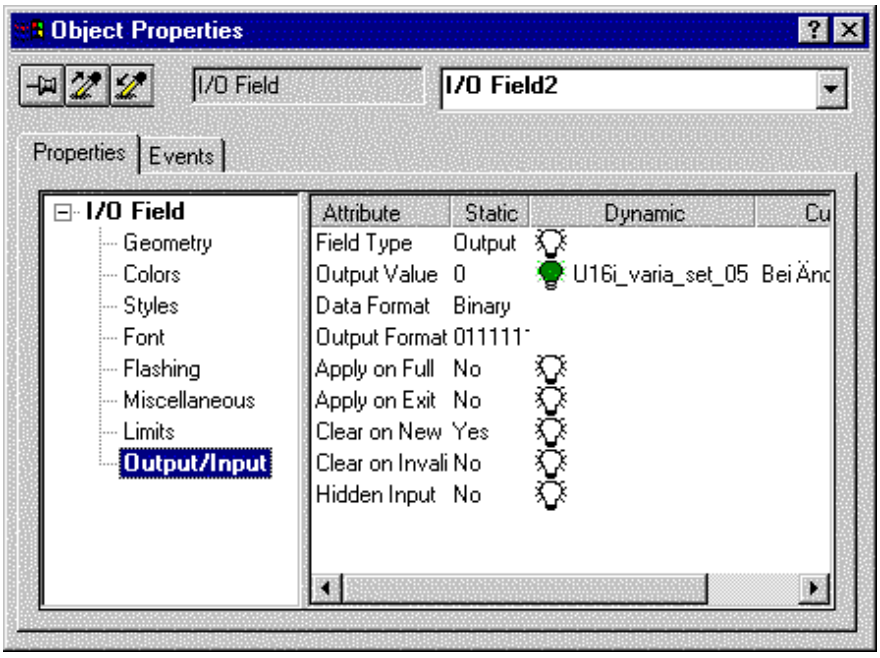
Постановка задачи

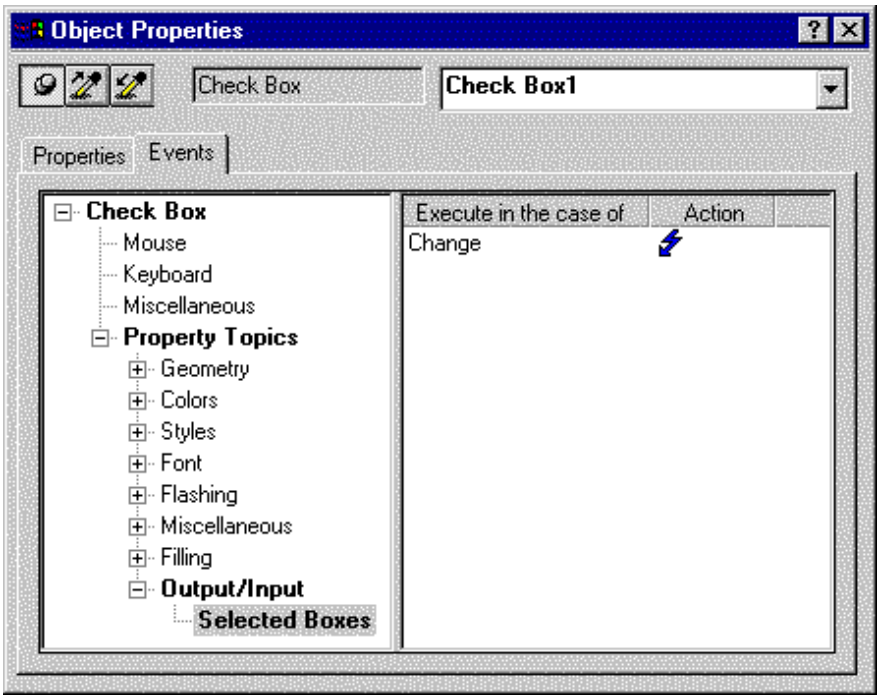
Необходимо изменять состояние выбранного бита в слове. Мы хотим иметь возможность выбирать несколько битов.

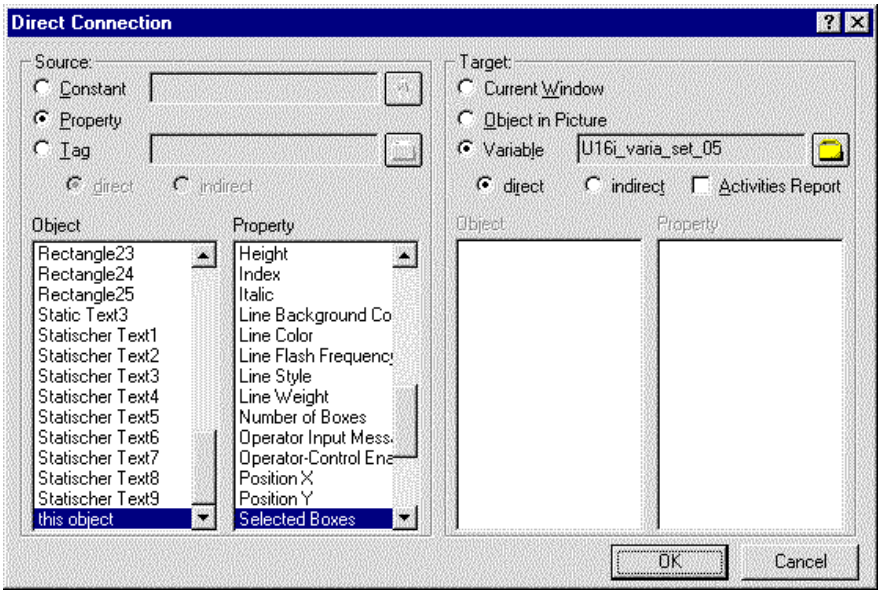
Концепция реализации

Для реализации изменения состояния бита мы воспользуется объектом *Windows Object (Объект Windows)* → *Check–Box (Набор флажков)*. Если одно из полей *набора флажков* выбрано с помощью (мыши), то связанный с ним бит изменяется во *внутреннем теге* посредством механизма *прямого соединения*. Для отображения состояния битов используется *Smart Object (Интеллектуальный объект)* → I/O Field *(Поле ввода/вывода)*. Выходное значение *поля ввода/вывода* связывается с *внутренним тегом.*

Реализация в проекте WinCC

Шаг	Процедура: Прямая установка бита при помощи флажков и прямого соединения
1	В менеджере тегов создайте тег типа *Unsigned 16–Bit Value (16–битная величина без знака)*. В данном примере используется тег *U16i_varia_set_05*.
2	Поместите в кадр *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В данном примере используется объект *I/O Field2*. При создании *поля ввода/вывода* в *диалоге конфигурирования* укажите тег *U16i_varia_set_05*. Поменяйте значение в поле *Update (Обновление)* на *Upon Change (По изменению)* и установите *Field Type (Тип поля)* в *Output (Вывод)*. В пункте *Properties (Свойства)* → *Output/Input (Вывод/ввод)*, измените *Data Format (Формат данных)* на *Binary (Двоичный)* и *Output Format (Формат вывода)* на *0111111111111111*. 

3	В том же кадре разместите *Windows Object (Объект Windows)* → *Check–Box (Набор флажков)*. В данном примере используется объект *Check–Box1*. В пункте *Properties (Свойства)* → *Geometry (Геометрия)* → *Number of Boxes (Количество флажков)*, поменяйте значение с 3 на *16*.
4	Выберите *Index 1* в *Properties (Свойства)* → *Font (Шрифт)* → *Index (Индекс)* → *1*. Для выбранного индекса введите соответствующий текст в пункте *Properties (Свойства)* → *Font (Шрифт)* → *Text* → *bit 0*. Аналогичным образом введите текст для других индексов.
5	Событие *Event (Событие)* → *Property Topics (Разделы свойств)* → *Selected Boxes (Выбранные флажки)* сделайте динамическим, используя *прямое соединение*. 

Шаг	Процедура: Прямая установка бита при помощи флажков и прямого соединения
6	В диалоге *Direct Connection (Прямое соединение)*, свяжите *Source Property (Источник, свойство)* → *this object (этот объект)* → *Selected Boxes (Выбранные флажки)* с *Target Variable(Приемник, переменная)* → *U16i_varia_set_05*. Подтвердите изменения нажатием на кнопку *OK*. 
7	Настройте два объекта типа *Windows Objects (Объекты Windows)* → *Buttons (Кнопка)*. В данном примере используются кнопки *Button2* и *Button3*. Они будут использованы для установки и сброса всех битов.
8	У кнопки *Button2* создайте *прямое соединение* для события *Event (Событие)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)*. Свяжите *source Constant (Источник, константа)* → *65535* с *target Object in Picture (Приемник, Объект кадра)* → *Check–Box1* → *Selected Boxes (Выбранные флажки)*. Подтвердите изменения нажатием на кнопку *OK*. Выбранная константа соответствует двоичному числу 1111111111111111. Для кнопки *Button3*, создайте *прямое соединение* таким же образом, но с *Source Constant (Источник, константа)* → *0*.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо проделать следующее:

В настройках *прямых соединений* указать нужные теги.

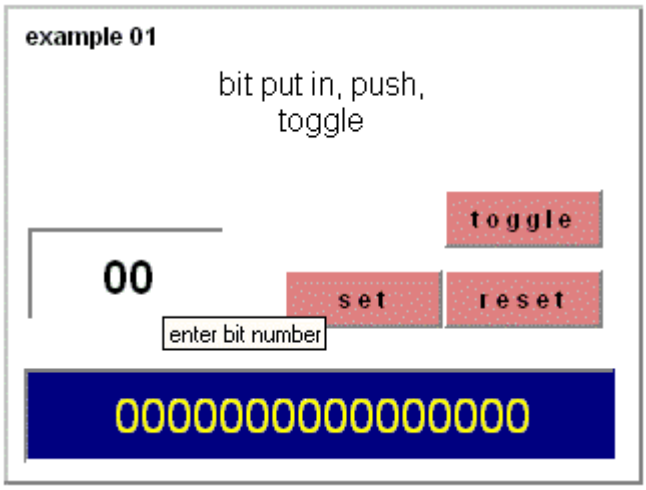
2.4.2 Выбор бита и изменение его состояния (example 01)

Постановка задачи

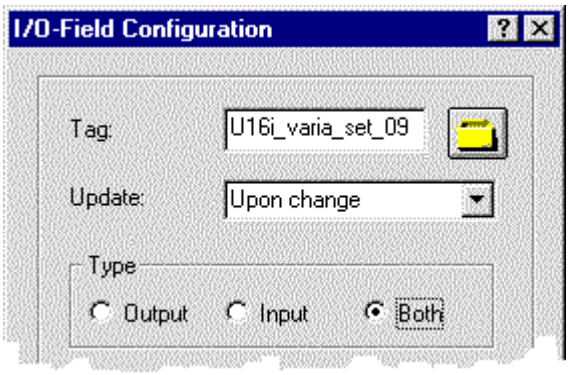
Организовать изменение состояния бита в слове по вводу его номера и нажатию на *кнопку*. Переключение должно производится с 0 на 1 и наоборот.

Концепция реализации

Для реализации изменения состояния бита используется *Windows Object (Объект Windows)* → *Button (Кнопка)*.
Для ввода номера бита и отображения его статуса используется *Smart Object (Интеллектуальный объект)* → I/O Field *(Поле ввода/вывода)*. При вводе номера бита и нажатии на *кнопку* (мышью) производится изменение выбранного бита во внутреннем теге. Это изменение реализовано с использованием *процедуры Си*.



Реализация в проекте WinCC

Шаг	Процедура: Изменение битов в слове
1	В менеджере тегов создайте два тега типа *Unsigned 16–Bit Value (16–битная величина без знака)*. В данном примере используются теги *U16i_varia_set_08* и *U16i_varia_set_09*.
2	Поместите в кадр *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В данном примере используется объект *I/O Field2*. При конфигурировании свяжите *поле ввода/вывода* с тегом *U16i_varia_set_09*. Поменяйте значение в поле *Update (Обновление)* на *Upon Change (По изменению)*. Ввод номера бита будет производиться в этом поле. 
3	Для отображения статуса бита поместите второе *I/O Field (Поле ввода/вывода)*. В данном примере используется объект *I/O Field1*. При его конфигурировании свяжите *поле ввода/вывода* с тегом *U16i_varia_set_08*. Поменяйте значение в поле *Update (Обновление)* на *Upon Change (По изменению)*. Смените тип поля на *Output (Вывод)*. В пункте *Properties (Свойства)* → *Output/Input (Вывод/ввод)*, поменяйте *Data Format (Формат данных)* на *Binary (Двоичный)* и *Output Format (Формат вывода)* на *0111111111111111*.
4	В том же кадре разместите три объекта типа *Windows Objects (Объект Windows)* → *Buttons (Кнопка)*. В данном примере используются кнопки *Button1*, *Button2* и *Button3*.
5	Для *Button1* создайте *процедуру Cи*, связанную с событием *Event (Событие)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Эта *процедура Cи* установит бит *внутреннего тега*, выбранный в *поле ввода/вывода*. Таким же образом создайте дополнительные *процедуры Cи* у других *кнопок* для сброса и переключения битов.

Процедура Си для кнопки установки (Set)

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
WORD word,pos;

//get word and bit position
pos=GetTagWord("U16i_varia_set_09");
word=GetTagWord("U16i_varia_set_08");

word = (WORD)(word|1<<pos);

SetTagWord("U16i_varia_set_08",word);
}
```

Объявляются *переменные Си*.

Для считывания введенного номера бита и текущего значения переменной используется *внутренняя функция GetTagWord*.

Используются операции побитового сдвига и сложения (ИЛИ).

Новое значение записывается во *внутренний тег*.

Процедура Си для кнопки сброса (Reset)

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
WORD word,pos;

//get word and bit position
pos=GetTagWord("U16i_varia_set_09");
word=GetTagWord("U16i_varia_set_08");

word=(WORD)(word&~(1<<pos));

SetTagWord("U16i_varia_set_08",word);
}
```

Объявляются *переменные Си*.

Для считывания введенного номера бита и текущего значения переменной используется *внутренняя функция GetTagWord*.

Используются операции побитового сдвига, инверсии, и умножения (И).

Новое значение записывается во *внутренний тег*.

Процедура Си для кнопки переключения (Toggle)

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
WORD word,pos;

//get word and bit position
pos=GetTagWord("U16i_varia_set_09");
word=GetTagWord("U16i_varia_set_08");

word = (WORD)(word^1<<pos);

SetTagWord("U16i_varia_set_08",word);
}
```

Объявляются *переменные Си*.

Для считывания введенного номера бита и текущего значения переменной используется *внутренняя функция GetTagWord*.

Используются операции побитового сдвига и исключающего ИЛИ.

Новое значение записывается во *внутренний тег*.

2.4.3 Остальные примеры главы

Example 02

Функционально этот пример похож на *example 01*. Основное отличие заключается в том, что переключаемый бит можно выбирать. В этом примере бит переключается при выборе объекта, представляющего этот бит. Каждый объект по своему имени имеет возможность определить за вывод состояния какого бита он отвечает.

Example 04

Функционально этот пример похож на *example 02*. Основное отличие заключается в том, что бит переключается сразу после того, как его выбрали  (мышью). Здесь тоже объекты ассоциируются с битами по именам.

Example 05

Функционально этот пример похож на *example 06*. Отличие состоит в том, что используется группа выбора *(радио–кнопка)*. Применение этого типа объектов дает возможность изменить только один бит в каждом слове.

2.5 Косвенная адресация тегов

Приемы, описываемые в данной главе, доступны для изучения в проекте *Project_TagHandling* по нажатию (мышью) на изображенной выше *кнопке*. Они приведены в кадре *varia_3_chapter_04.pdl*.

2.5.1 Косвенная адресация при помощи прямого соединения (example 01)

Постановка задачи

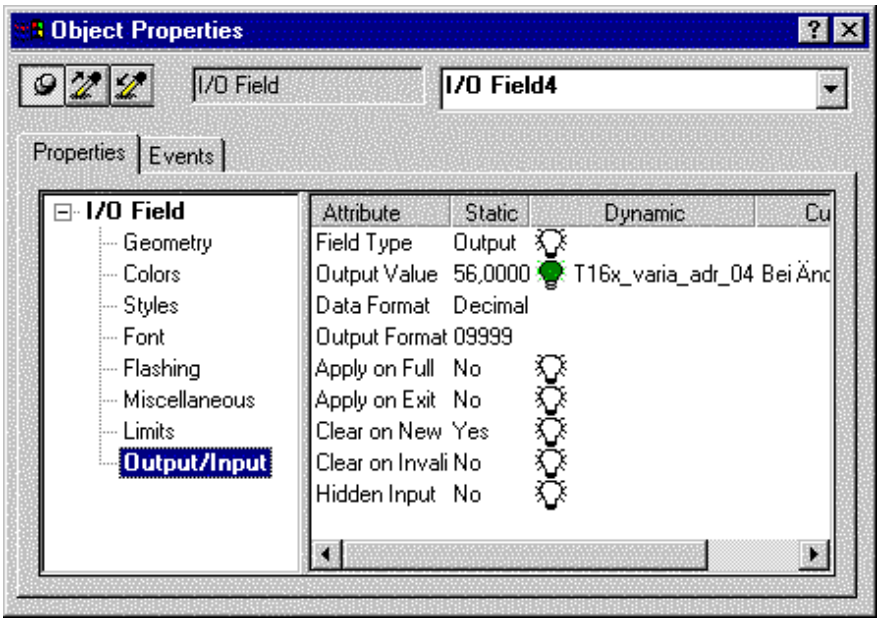
Отобразить в *поле ввода/вывода* параметры процесса. Выбор отображаемых параметров должен осуществляться при помощи *кнопок*.

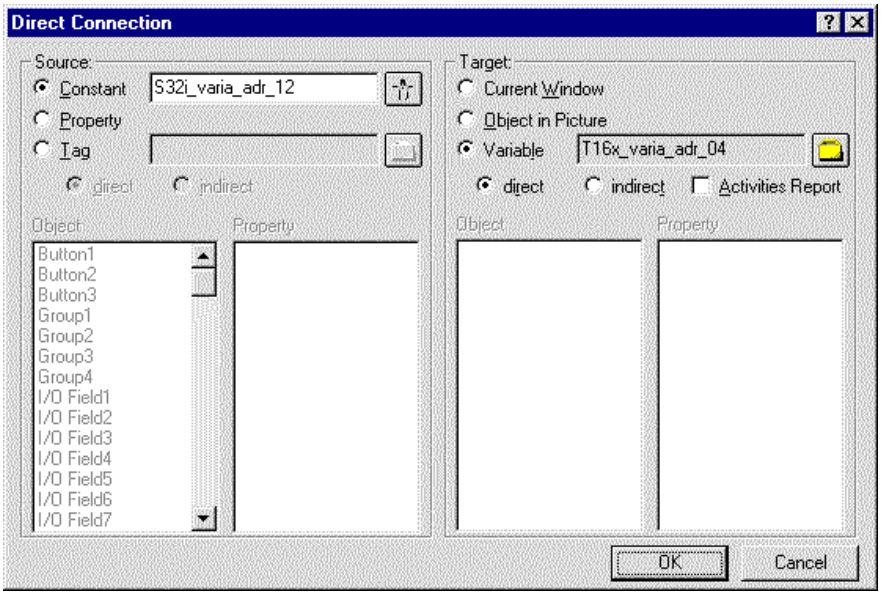
Концепция реализации

Для реализации выбора отображаемых параметров процесса мы будем использовать объект типа *Windows Object (Объект Windows)* → *Button (Кнопка)*.
Для отображения параметров процесса выберем *Smart Object (Интеллектуальный объект)* → I/O Field *(Поле ввода/вывода)* и метод косвенной адресации WinCC. Три дополнительных *Smart Objects (Интеллектуальный объект)* → I/O Fields *(Поля ввода/вывода)* создаются для организации прямого ввода параметров процесса.

Реализация в проекте WinCC

Шаг	Процедура: Косвенная адресация при помощи прямого соединения
1	В менеджере тегов создайте три тега типа *Signed 32–Bit Value (32–битная величина со знаком)*. В данном примере используются теги *S32i_varia_adr_12*, *S32i_varia_adr_13* и *S32i_varia_adr_14*. В них хранятся параметры процесса, которые необходимо отобразить.
2	В менеджере тегов создайте тег типа *Text Tag 16–Bit Character Set (Текстовый тег 16–битного набора символов)*. В данном примере используется тег *T16x_varia_adr_04*. Этот тег будет использован в качестве адресного тега.

3	Поместите в кадр *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В данном примере используется объект *I/O Field4*. При создании *поля ввода/вывода* в *диалоге конфигурирования* укажите тег *T16x_varia_adr_04*. Смените установленное по умолчанию значение в поле *Update (Обновление)* с 2 s на *Upon Change (По изменению)* и установите *Field Type (Тип поля)* в *Output (Вывод)*. Для атрибута *Properties (Свойства)* → *Output/Input (Вывод/ввод)* → *Output Value (Выходное значение)* активизируйте флажок в колонке *Indirect (Косвенная адресация)*. 
4	В том же кадре разместите три дополнительных *поля ввода/вывода*. В данном примере используются объекты с *IO–Field1* по *IO–Field3*. При создании *IO–Field1* в *диалоге конфигурирования* укажите тег *S32i_varia_adr_12* и триггер *Upon Change (По обновлению)*. Таким же образом настройте *поля ввода/вывода 2* и *3*, но свяжите их с другими адресными тегами.
5	Поместите в кадр объект типа *Standard Object (Стандартные объекты)* → *Static Text (Статический текст)*. В данном примере используется объект *Static Text1*. Этот объект показывает, какой параметр процесса отображается в данный момент. Текст автоматически заносится в объект при помощи *кнопки*.
6	В том же кадре разместите три объекта *Windows Objects (Объект Windows)* → *Buttons (Кнопки)*. В данном примере используются объекты *Button1*, *Button2* и *Button3*.

7	У *Button1* настройте *прямое соединение* для события *Event (Событие)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Свяжите *Source Constant (Источник, константа)* → *S32i_varia_adr_12* с *Target Variable (Приемник, переменная)* → *T16x_varia_adr_04*. Подтвердите изменения нажатием на кнопку *OK*. 
8	Создайте другое *прямое соединение* для события *Event (Событие)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)*. Свяжите *Source Property (Источник, свойство)* → *this object (этот объект)* → *Text (Текст)* с *Target Object in Picture (Приемник, объект кадра)* → *Static Text1* → *Text (Текст)*. Подтвердите изменения нажатием на кнопку *OK*.
9	Настройте кнопки *Button2* и *Button3* таким же образом, как и *Button1*. Для первого *прямого соединения* имя тега *источника* необходимо изменить. Второе *прямое соединение* можно использовать без каких–либо изменений.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо проделать следующее:

Указать названия нужных тегов.

2.5.2 Множественное отображение посредством косвенной адресации и процедуры Си (example 02)

Постановка задачи

Отобразить три различных параметра контейнера. Необходимо иметь возможность использовать копии одного и того же визуального объекта для отображения параметров различных контейнеров. При выборе контейнера следует отображать его параметры.

Концепция реализации

Для реализации выбора необходимого контейнера используется *Windows Object (Объект Windows)* → *Option Group (Группа выбора)*. Для отображения параметров используются *Smart Objects (Интеллектуальные объекты)* → *I/O Fields (Поля ввода/вывода)* и механизм косвенной адресации WinCC. Контейнеры с соответствующими параметрами отображаются в *example 04*.

Реализация в проекте WinCC

Шаг	Процедура: Множественное отображение посредством косвенной адресации
1	В менеджере тегов создайте девять тегов типа *Signed 32–Bit Value (32–битная величина со знаком)*. В данном примере используются теги с *S32i_varia_adr_03* по *S32i_varia_adr_11*. В них хранятся соответствующие параметры контейнеров.
2	В менеджере тегов создайте три тега типа *Text Tag 16–Bit Character Set (Текстовый тег 16–битного набора символов)*. В данном примере используются теги *T16x_varia_adr_01*, *T16x_varia_adr_02* и *T16x_varia_adr_03*. Эти теги будут использованы для адресации *полей ввода/вывода*.
3	Настройте три *Smart Objects (Интеллектуальные объекты)* → *I/O Fields (Поля ввода/вывода)*. В данном примере используются объекты *I/O Field5*, *I/O Field6* и *I/O Field7*.
4	При создании *I/O Field5* в *диалоге конфигурирования* укажите тег *T16x_varia_adr_01*. Смените значение поля *Update (Обновление)* на *Upon Change (По изменению)* и *Field Type (Тип поля)* на *Output (Вывод)*. У атрибута *Properties (Свойства)* → *Output/Input (Вывод/ввод)* → *Output Value (Выходное значение)* активизируйте флажок в колонке *Indirect (Косвенная адресация)*.
5	Аналогичным образом настройте остальные *поля ввода/вывода*, но при этом свяжите каждое из них с собственным адресным тегом.
6	Поместите в кадр *Windows Object (Объект Windows)* → *Option Group (Группу выбора)*. В данном примере используется объект *Option Group1*.
7	Выберите индекс 1 в поле *Properties (Свойства)* → *Font (Шрифт)* → *Index (Индекс)*. Для выбранного индекса введите соответствующий текст у атрибута *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* → *Container1*. Аналогичным образом задайте текст для других индексов.
8	Для события *Events (События)* → *Property Topics (Разделы свойств)* → *Output/Input (Вывод/ввод)* → *Selected Boxes (Выбранные пункты)* создайте *процедуру Си*. Эта процедура будет записывать в адресный тег значение, соответствующее выбранному полю.

Процедура Си для группы выбора

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l

{
char address1[20],address2[20],address3[20];

switch(value) {
        case 2: {
                strcpy(address1,"S32i_varia_adr_03");
                strcpy(address2,"S32i_varia_adr_06");
                strcpy(address3,"S32i_varia_adr_09");
                }//case 2
                break;
        case 4: {
                strcpy(address1,"S32i_varia_adr_04");
                strcpy(address2,"S32i_varia_adr_07");
                strcpy(address3,"S32i_varia_adr_10");
                }//case 4
                break;
        default: {
                strcpy(address1,"S32i_varia_adr_05");
                strcpy(address2,"S32i_varia_adr_08");
                strcpy(address3,"S32i_varia_adr_11");
                }//default
                break;
        }//switch

SetTagChar("T16x_varia_adr_01",address1);
SetTagChar("T16x_varia_adr_02",address2);
SetTagChar("T16x_varia_adr_03",address3);
}
```

Объявляются три *переменные Си* — символьные массивы.

Имена переменных копируются в соответствии с входными значениями в ранее объявленные теги. Состояние входных параметров хранится в предварительно описанной переменной *value*.

Соответствующие имена переменных заносятся в адресные теги.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо проделать следующее:

Указать названия нужных тегов.

2.5.3 Косвенная адресация посредством процедуры Си (example 03)

Постановка задачи

В *поле ввода/вывода* необходимо отображать параметры процесса.
Отображаемые параметры должны выбираться с использованием *группы выбора*.

Концепция реализации

Для реализации выбора отображаемых параметров процесса используется *Windows Object (Объект Windows)* → *Option Group (Группа выбора)*. Для отображения параметров используется *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)* и механизм косвенной адресации WinCC.

Исполнение в проекте WinCC

Шаг	Процедура: Косвенная адресация посредством процедуры Си
1	В менеджере тегов создайте три тега типа *Signed 32–Bit Value (32–битная величина со знаком)*. В данном примере используются теги *S32i_varia_adr_00*, *S32i_varia_adr_01* и *S32i_varia_adr_02*. В них хранятся параметры процесса, которые необходимо отобразить.
2	В менеджере тегов создайте тег типа *Text Tag 16–Bit Character Set (Текстовый тег 16–битного набора символов)*. В данном примере используется тег *T16x_varia_adr_00*. Он будет использован в качестве адресного тега.
3	Поместите в кадр *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В данном примере используется объект *I/O Field8*. При создании *поля ввода/вывода* в *диалоге конфигурирования* укажите тег *T16x_varia_adr_00*. Смените значение в поле *Update (Обновление)* на *Upon Change (По изменению)* и установите *Field Type (Тип поля)* в *Output (Вывод)*. У атрибута *Properties (Свойства)* → *Output/Input (Вывод/ввод)* → *Output Value (Выходное значение)* установите флажок в колонке *Indirect (Косвенная адресация)*.
4	В том же кадре разместите *Windows Object (Объекты Windows)* → *Option Group (Группу выбора)*. В данном примере используется объект *Option Group2*.
5	Выберите индекс 1 в поле *Properties (Свойства)* → *Font (Шрифт)* → *Index (Индекс)*. Для выбранного индекса введите соответствующий текст в элементе *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* → *Fill Level (Уровень заполнения)*. Аналогичным образом задайте текст для других значений индекса.
6	Для события *Events (События)* → *Property Topics (Разделы свойств)* → *Output/Input (Вывод/ввод)* → *Selected Boxes (Выбранные пункты)* создайте *процедуру Си*. Эта процедура будет записывать в адресный тег значение, соответствующее выбранному полю.

Процедура Си для группы выбора

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l

{
char address[40];

//set tag according to input value
switch(value) {
   case 2: strcpy(address,"S32i_varia_adr_01");
   break;
   case 4: strcpy(address,"S32i_varia_adr_02");
   break;
   default: strcpy(address,"S32i_varia_adr_00");
   }//switch
SetTagChar("T16x_varia_adr_00",address);
}
```

В адресную переменную *T16x_varia_adr_00* заносятся имена тегов, соответствующие состоянию входных параметров. Состояние входных параметров хранится в предварительно описанной переменной *value*..

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо проделать следующее:

Указать названия нужных тегов.

2.5.4 Остальные примеры главы

Example 04

Этот пример должен отображать значения параметров процесса, используемые в *example 02*.

2.6 Моделирование изменения значений тегов

Приемы, описываемые в данной главе, доступны для изучения в проекте *Project_TagHandling* по нажатию (мышью) на изображенной выше *кнопке*. Они приведены в кадре *varia_3_chapter_05.pdl*.

Определение

Термин моделирование относится к изменению содержимого тега без соединения с драйвером процесса. Моделирование производится с помощью специальной утилиты.

2.6.1 Моделирование пилообразного сигнала при помощи процедуры Си (example 01)

Постановка задачи

Необходимо смоделировать пилообразный сигнал с изменяемыми минимальным и максимальным значениями. Корректность задаваемых значений необходимо проверять при вводе. Моделирование нужно запускать и останавливать по нажатию *кнопки*. Другая *кнопка* должна использоваться для сброса значения тега в ноль.

Концепция реализации

Для реализации запуска/останова и инициализации моделирования используются две *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. Для отображения значения тега и ввода максимального и минимального значений используются *Smart Objects (Интеллектуальный объект)* → *I/O Fields (Поля ввода/вывода)*. При попытке запуска моделирования с одинаковыми значениями минимума и максимума, будет выведено сообщение.

Реализация в проекте WinCC

Шаг	Процедура: Моделирование пилообразного сигнала с помощью процедуры Си
1	В менеджере тегов создайте три тега типа *Signed 32–Bit Value (32–битная величина со знаком)*. В данном примере используются теги *S32i_varia_sim_00*, *S32i_varia_sim_02* и *S32i_varia_sim_03*.
2	Создайте два тега типа *Binary Tag (Двоичный тег)*. В данном примере используются теги *BINi_varia_sim_01* и *BINi_varia_sim_04*.
3	Поместите в кадр три *Smart Objects (Интеллектуальный объект)* → *I/O Fields (Поля ввода/вывода)*. В данном примере используются объекты *I/O Field1*, *I/O Field2* и *I/O Field3*.
4	При создании *I/O Field1* в *диалоге конфигурирования* укажите тег *S32i_varia_sim_03* и триггер *Upon Change (По изменению)*. В поле *Properties (Свойства)* → *Output Format (Формате вывода)* измените формат на *0999*. Аналогичным образом настройте *I/O Field2*, но для него укажите тег *S32i_varia_sim_02*.
5	Для проверки корректности значения *I/O Field1* установите *Tag Connection (Связь с тегом)* для тега *S32i_varia_sim_02* и поля *Properties (Свойства)* → *Limits (Ограничения)* → *High Limit Value (Значение верхней границы)*. Аналогичным образом укажите тег *S32i_varia_sim_03* в качестве атрибута *Low Limit Value (Значение нижней границы)* для *I/O Field2*.
6	При создании объекта *I/O Field3* в *диалоге конфигурирования* укажите тег *S32i_varia_sim_00*, триггер *Upon Change (По изменению)* и *Field Type (Тип поля) Output (Вывод)*. В атрибуте *Properties (Свойства)* → *Output/Input (Вывод/ввод)* → *Output Format (Формат вывода)* измените формат на *0999*.

7	Настройте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, в данном примере это *dialog box (диалоговое окно)*. В поле *Properties (Свойства)* → *Miscellaneous (Различные)*, измените свойство *Moveable (Перемещаемое)* и *Border (Рамка)* на *Yes (Да)* и *Picture Name (Имя кадра)* на *varia_5_window_00*. Этот кадр можно скопировать из проекта примера для использования в вашем собственном проекте; информационный текст и заголовок можно изменить в соответствии с вашими требованиями.
8	Настройте объект *Windows Object (Объект Windows)* → *Button (Кнопка)*, в данном примере это *Button2*. Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*, создайте *прямое соединение*. Свяжите *Source Constant (Источник, константа)* → *1* с *Target Variable (Приемник, переменная)* → *BINi_varia_sim_04*. Эта *кнопка* используется для инициализации.
9	Настройте другой объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется кнопка *Button1*. Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*, создайте *процедуру Си,* которая инвертирует тег *BINi_varia_sim_01*. Для атрибута *Properties (Свойства)* → *Geometry (Геометрия)* → *Position X (Координата X)* создайте *процедуру Си*, которая будет выполнять моделирование.
10	Для отображения процесса моделирования настройте *Smart Object (Интеллектуальный объект)* → *Status Display (Окно состояния)*. В данном примере используется *Status Display1*. В *диалоге конфигурирования* укажите тег *BINi_varia_sim_01* и триггер *Upon Change (По изменению)*. Добавьте еще одно окно состояния. Для состояния *0* назначьте рисунок *glühbirne_2_24Bit.gif*, а для состояния *1* — рисунок *glühbirne_1_24Bit.gif*.

Процедура Си для моделирования тегов

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyName)
{
BOOL state;
static DWORD lowstore = 0;
static BOOL statestore = 0;
static DWORD r = 1;
static DWORD i = 0;
static DWORD box = 0;
int high,low;

//if button init was pressed
if (GetTagBit("BINi_varia_sim_04")) {
      (i=lowstore);
      (r=1);
      SetTagDWord("S32i_varia_sim_00",i);
      SetTagDWord("BINi_varia_sim_04",0);
       }

//get simulator state
state=GetTagBit("BINi_varia_sim_01");

if (state!=statestore) (box=0);

statestore=state;

//get limits
high=GetTagDWord("S32i_varia_sim_02");
low=GetTagDWord("S32i_varia_sim_03");

//set low limit store
if (low!=lowstore){
        lowstore = low;
        i=low;
        }//if

//if limits different
if (high!=low) {
        box=0;
        //if simulator is activated
        if (state==TRUE) {
     //inc or dec according to direction
        if (r==1) (i=i+1);
        else (i=i-1);
        //set direction
        if (i==high) (r=0);
        if (i==low) (r=1);
        //init simulator if limit overflow
        if ((i>high)||(i<low)){
                (i=low);
                (r=1);
                }//if
        //set new value
        SetTagDWord("S32i_varia_sim_00",i);
        }//if state
}//if (high!=low)
//set visible message box
if ((high==low)&&(state==1)&&(box==0)){
        box++;
        SetVisible("varia_3_chapter_05.PDL","dialog box",1);
        }
return 80;//x-pos
}
```

Производится объявление тегов.

Если кнопка *Button2* (инициализационная) нажата, счетчик изменяется на возрастание, во *внутренний тег S32i_varia_sim_00* заносится сохраненное значение минимума, и моделирование отключается.

Считывается состояние моделирования.

Если состояние изменилось, выводится окно сообщения.

Сохраняется состояние.

Считываются максимальное и минимальное значения.

Если изменилось минимальное значение, обновляется значение соответствующей переменной.

Если максимум и минимум различны, выводится окно сообщения и, если процесс запущен, производится моделирование. Счетчик увеличивается или уменьшается в соответствии со значением тега; при достижении граничных значений производится реверс счетчика; если граничные значения превышены, производится инициализация и тег *S32i_varia_sim_00* устанавливается в минимальное значение.

Если моделирование запущено, отображение окна разрешено, и минимум и максимум согласуются друг с другом, окно сообщений становится видимым.

Возвращается значение, которое является координатой X кнопки *Button1*.

2.6.2 Моделирование с помощью внешней программы (example 02)

WinCC предоставляет собственную программу имитации, которая может моделировать поведение тегов различными способами. Этот имитатор должен быть установлен при помощи программы Setup.exe из папки *SmartTools* → *CC_Simulator* на WinCC CD–ROM.

Постановка задачи

Необходимо произвести имитацию изменения тегов с использованием имитатора WinCC.

Концепция реализации

Для реализации мы воспользуемся несколькими тегами, содержимое которых будет отображаться в объектах типа *Smart Objects (Интеллектуальный объект)* → *I/O Fields (Поля ввода/вывода)*. Значения этих тегов будут задаваться имитатором.

Реализация в проекте WinCC

Шаг	Процедура: Моделирование с помощью внешней программы
1	В менеджере тегов создайте два *внутренних тега* типа *Signed 32–Bit Value (32–битная величина со знаком)*. В данном примере используются теги *S32i_varia_sim_05* и *S32i_varia_sim_06*.
2	Настройте два объекта типа *Smart Objects (Интеллектуальные объекты)* → *I/O Fields (Поля ввода вывода)*. В данном примере используются объекты *I/O Field4* и *I/O Field5*.
3	При создании *I/O Field4* в *диалоге конфигурирования* укажите тег *S32i_varia_sim_05*, триггер *Upon change (По изменению)* и установите *Field Type (Тип поля)* в *Output (Вывод)*. В поле *Properties (Свойства)* → *Output/Input (Вывод/ввод)* измените *Output Format (Формат вывода)* на *0999.999*. Точно так же выполните настройку для *I/O Field5*, но назначьте тег *S32i_varia_sim_06*.
4	Запустите имитатор по щелчку на кнопку *Simulator (Имитатор)*. Имитатор тегов запускается посредством *процедуры Cu*, созданной для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Если имитатор тегов установлен в нестандартном месте, укажите к нему правильный путь, используя кнопку *Path*. Если имитатор был запущен каким–либо другим образом, следует убедиться, что рассматриваемый проект находится в режиме исполнения.
5	В окне имитатора выберите тег *S32i_varia_sim_05* из менеджера тегов, используя пункт меню *Edit (Редактирование)* → *New Tag (Новый тег)*. Выберите закладку *Inc (Возрастание)* и введите *Start Value (Начальное значение)* и *End Value (Конечное значение)*. В данном примере используются значения *0* и *20*. Моделирование запускается командой меню *Active (Активизировать)*. Значение тега увеличивается от 0 до 20, после чего моделирование перезапускается с 0.

6	Произведите описанные действия с тегом *S32i_varia_sim_06*. В данном примере была выбрана закладка *Sinus (Синус)*: *Amplitude (Амплитуда)* установлена в *50*, *Offset (Смещение)* в *50* и *Oscillation Time (Период колебания)* в *25*.

Процедура Си для кнопки моделирования

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpsz
{
char simdeupath[200];
char program[15];
int result;

if (GetTagBit("BINi_varia_sim_10")) strcpy(program,"\\simeng.exe");
else strcpy(program,"\\simdeu.exe");

//build path to simdeu
strcpy(simdeupath,GetTagChar("T16x_varia_sim_07"));
strcat(simdeupath,program);

//execute simdeu
result=ProgramExecute(simdeupath);

//if not able to execute simdeu set visible message box
if (result<32) {
      SetVisible("varia_3_chapter_05.PDL","error message",1);
      }
}
```

Производится объявление тегов.

Если переменная *BINi_varia_sim_10* установлена, имя английской версии имитатора записывается в переменную программы. В противном случае туда записывается имя немецкой версии программы.

С использованием встроенной функции *GetTagChar* считывается путь к файлу имитатора.

К пути добавляется стартовый файл.

Запускается имитатор.

Если путь ошибочен, выводится сообщение об ошибке.

Примечание:

В проекте примера для выбора английской или немецкой версии имитатора можно использовать иконку флага.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо проделать следующее:

Моделируемые теги и метод моделирования должны быть выбраны в соответствии с вашими требованиями.

2.7 Импорт / экспорт тегов

Приемы, описываемые в данной главе, доступны для изучения в проекте *Project_TagHandling* по нажатию (мышью) на изображенной выше *кнопке*. Они приведены в кадре *varia_3_chapter_06.pdl*.

Постановка задачи

Содержимое менеджера тегов необходимо прочитать какой–либо программой и отредактировать в MS Excel (электронной таблице). Необходимо иметь возможность импортировать модифицированные данные обратно в проект WinCC. Эта процедура позволяет создавать большое количество тегов без значительных усилий.

Концепция реализации

В реализации используются два объекта типа *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*, применяемые для запуска программы импорта/экспорта *var_imex.exe* и редактора электронных таблиц *excel.exe*. Путь к каждой из этих программ может быть задан при помощи двух объектов *Smart Objects (Интеллектуальные объекты)* → *I/O Fields (Поля ввода/вывода)*.

Реализация в проекте WinCC

Шаг	Процедура: Импорт / экспорт тегов
1	Задайте путь к программам *excel.exe* и *var_imex.exe*.
2	Запустите *var_exim.exe* нажатием на *кнопку Imp/Exp* в режиме исполнения. Программа может быть запущена также прямо из проводника Windows, при этом WinCC не обязательно должна быть в режиме исполнения.
3	С помощью *кнопки* ..... укажите путь к проекту *Project_TagHandling* и выберите файл *Project_TagHandling.mcp*.
4	В поле выбора укажите *Export (Экспорт)*. Процедура Си для вызова внешней программы описана в примере 1.6.2. Затем нажмите *Execute (Исполнить)* → *OK*. После этого запустится процесс экспорта тегов. Программа генерирует файл с расширением *vex*, содержащий информацию о тегах, второй файл с расширением *cex* описывающий связи с PLC, и третий файл с расширением *dex*, содержащий информацию о тегах типа *Data Structure (Структура данных)*.
5	Запустите *Excel* и откройте только что созданный файл *Project_vex.csv* посредством команды *File (Файл)* → *Open (Открыть)*.
6	Для настройки 100 тегов типа *unsigned 16–Bit value (16–битная величина без знака)*, выполните следующее. В качестве имен тегов используйте *U16i_varia_impex_00* – *U16i_varia_impex_99*.

7	В первой пустой строке первой колонки введите имя *U16i_varia_impex_00*. Выберите ячейку и переместите указатель мыши в ее правый нижний угол. Удерживая нажатой (мышь), переместите указатель вниз для автоматического заполнения 99 ячеек.
8	Во второй колонке введите *; в третей введите *Internal Tag (Внутренний тег)*; в четвертой в качестве имени группы введите *impexp*; в пятой введите *2*, в шестой в качестве кода поля для *Unsigned 16–Bit Value (16–битной величины без знака)* введите *5*. В других колонках введите значение *0*. Заполните остальные 99 строк автоматически.
9	Через панель задач вновь откройте *Var_imex.exe* и выберите поле *Import (Импорт)*. Затем нажмите *Execute (Исполнить)* → *OK*. После завершения импорта тегов закройте программу.
10	Теперь в менеджере тегов создано 100 новых тегов.

Примечание:

При импорте и экспорте тегов режим исполнения не должен быть запущен.

2.8 Использование структурных тегов

Приемы, описываемые в данной главе, доступны для изучения в проекте *Project_TagHandling* по нажатию (мышью) на изображенной выше *кнопке*. Они приведены в кадре *varia_3_chapter_07.pdl*.

Определение

Этот тип данных позволяет создавать структуры данных, образующие логические единицы. Структурные теги составляются из стандартных типов данных.

2.8.1 Управление клапаном с помощью структурного тега (example 01)

Постановка задачи

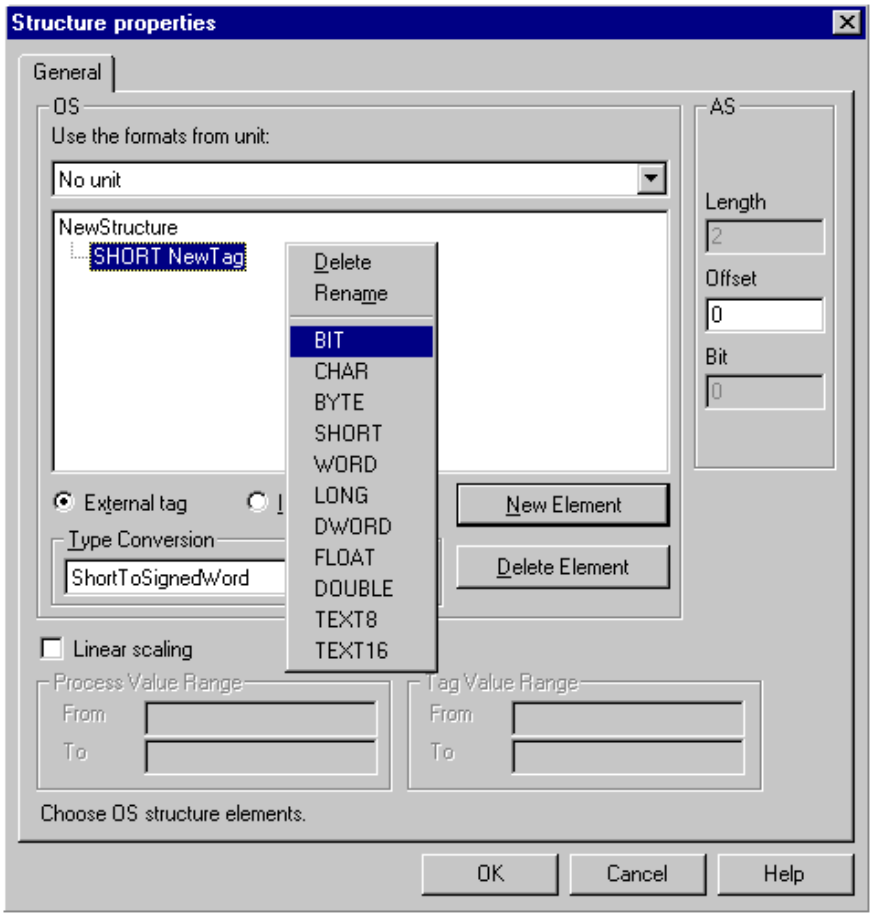
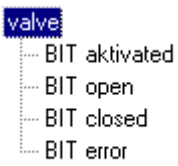
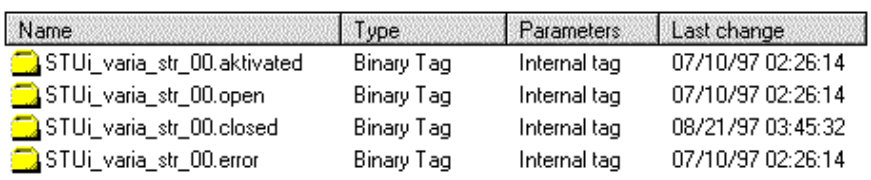
Организовать отображение различных состояний клапана с использованием структурных тегов.

Концепция реализации

В реализации мы используем два объекта типа *Windows Objects (Объекта окна)* → *Buttons (Кнопки)*, при помощи которых производится включение и выключение клапана, а также моделируется его неисправность. Для отображения клапана мы используем *Standard Objects (Стандартные объекты)* → *Polygons (Многоугольники)*.

Реализация в проекте WinCC

Шаг	Процедура: Управление клапаном с помощью структурного тега
1	Определите новый структурный тег в *проводнике WinCC*. На элементе *Data Types (Типы данных)* → *Structure Types (Структурные типы)* R (правой кнопкой мыши) выберите пункт *New Structure (Новая структура)* во всплывающем меню. 

Шаг	Процедура: Управление клапаном с помощью структурного тега
2	В следующем окне нажмите R (правой кнопкой мыши) на элемент *New Structure (Новая структура)* выберите *Rename (Переименовать)* во всплывающем меню. В данном примере используется имя *valve*. При помощи кнопки *New Element (Новый элемент)* добавьте новый структурный элемент. Используя R (правую кнопку мыши) на вновь созданном элементе установите его тип данных в *Bit (Битовый)*. 
3	Используя кнопку *Rename (Переименовать)*, смените имя элемента на *activated* и выберите радио кнопку *internal (внутренний)*. Задайте остальные элементы структуры следующим образом: 
4	В менеджере тегов создайте тег типа *valve*. В данном примере используется тег *STUi_varia_str_00*. При этом будут созданы следующие *двоичные теги*. 

Шаг	Процедура: Управление клапаном с помощью структурного тега
5	Настройте два объекта типа *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*, в данном примере используются *Button1* и *Button2*. Для события *Button1* → *Event (Событие)* → *Mouse Action (Действие мыши)* → *Press Left (Нажатие левой кнопки)*, создайте *процедуру Cи*, которая будет включать и выключать клапан. Аналогичным образом создайте *процедуру Cи* для кнопки *Button2*, которая будет устанавливать и сбрасывать бит ошибки.
6	У кнопки *Button1* создайте *процедуру Cи* для атрибута *Properties (Свойства)* → *Geometry (Геометрия)* → *Position X (Координата X)*, которая будет моделировать внешние по отношению к клапану процессы.
7	Затем создайте три изображения, соответствующие включенному, отключенному и ошибочному состояниям клапана. В примере эти изображения содержат два *Standard Objects (Стандартные объекты)* → *Polygons (Многоугольника)*. Они расположены один над другим и отображены или скрыты в зависимости от состояния клапана.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо проделать следующее:

Указать подходящее имя структурного типа и названия его составных элементов.

Процедуры Cи для моделирования внешних процессов клапана не нужны в реальных приложениях.

3 Конфигурация кадров (Project_CreatePicture)

Проект, созданный в этой главе, можно скопировать непосредственно из online–документа на ваш жесткий диск. По умолчанию он будет записан в папку *C:\Configuration_Manual*. .

 Project_CreatePicture

Детали проекта

Данный проект представляет различные способы структурирования и открытия кадров в WinCC. Структура кадра и процедура ее открытия зависят от двух факторов: от используемого аппаратного обеспечения (промышленного компьютера в форме панели оператора с интегрированной клавиатурой – OP47, или компьютера в зале управления с мышью и стандартной клавиатурой), и от приложения. К примеру, требования, предъявляемые к системе HMI машиностроительной и химической компаниями, могут существенно отличаться.

Какие возможности предлагает WinCC?

WinCC поддерживает все разрешения экрана, предусмотренные в Windows (т.е. 640x480, 800x600, 1024x768, 1280x1024). Иногда обзорные кадры должны отображаться с более высоким базовым разрешением (т.е. 1600x1028, 2000x1500, и т.д.).
WinCC позволяет вам создавать кадры с максимальным разрешением 4096x4096. Если эти параметры превышают максимальное разрешение используемой графической системы (видеоадаптера и монитора), кадры могут прокручиваться с использованием полос прокрутки.

Замечание

Предполагается, что разрешение графической системы установлено в 1024x768. Это разрешение соответствует рекомендациям с учетом эргономики для системы с 17–дюймовым монитором.

Примеры для данного раздела приведены в проекте WinCC *Project_CreatePicture*.

Примечание:

Пароль для входа — pictu_00. Просто щелкните на слове *Login (Регистрация)* в заголовке окна, введите пароль в поле ввода *Password (Пароль)*, после чего подтвердите ввод.

3.1 Макет экранной формы и смена кадра

В данной главе демонстрируется ряд различных способов структурирования и открытия кадров. Базовые элементы (стартовый кадр, обзорная секция и секция кнопок) макета экранной формы используются и в других проектах.

3.1.1 Макет экранной формы

Постановка задачи

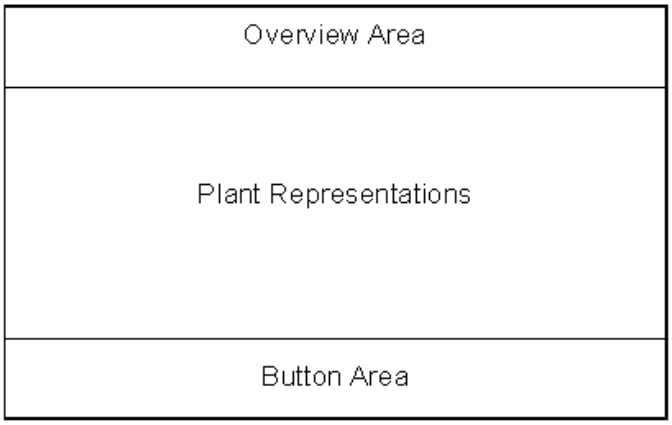
Динамический набор кнопок и секция обзора

Экран должен быть разделен на три секции:
секция обзора, секция кнопок и секция кадров оборудования.
Секции обзора и кнопок должны быть настраиваемыми.
Система расположена в операторском зале и управляется мышью и клавиатурой.

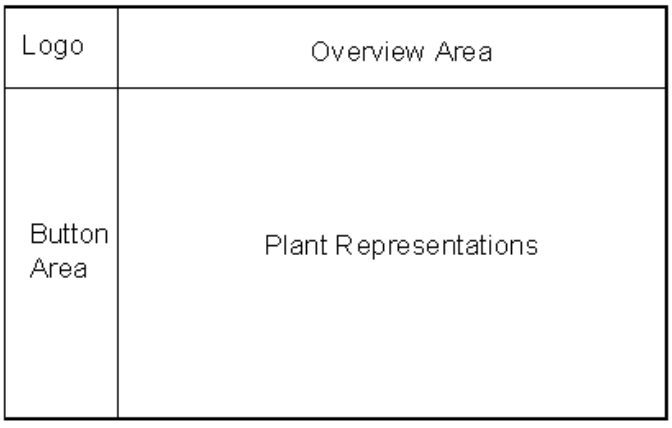
Концепция реализации

Разрешение экрана установлено в 1024x768. Мы разделим экранную форму на три секции. Будем использовать два различных макета для этих трех секций.

Макет 1



Макет 2



Принцип макетирования

Мы используем пустой стартовый кадр, в котором затем создадим 3 окна (обзор, кнопки, оборудование). Кадры, отображаемые в этих окнах, могут при необходимости переключаться во время выполнения. Это дает нам гибкое и легко модифицируемое решение.

Секция обзора

В секции обзора мы создаем логотип, заголовок кадра, часы с датой и временем, и строку аварийных сообщений.

Секция кнопок

В секции кнопок мы создаем постоянные кнопки, которые будут отображаться во всех кадрах, и кнопки, которые будут отображаться в зависимости от типа кадра оборудования.

Секция оборудования

В секции оборудования мы создаем соответствующие кадры оборудования.

3.2 Смена кадра

В режиме исполнения примеры, имеющие отношение к этой теме, доступны в проекте *Project_CreatePicture* по нажатию (мышью) на *кнопке*, изображенной выше. Примеры приведены в кадрах *pictu_3_chapter_01.pdl* и *pictu_3_chapter_01a.pdl*.

3.2.1 Открытие кадра с помощью прямого соединения и отображение его названия (example 01)

Постановка задачи

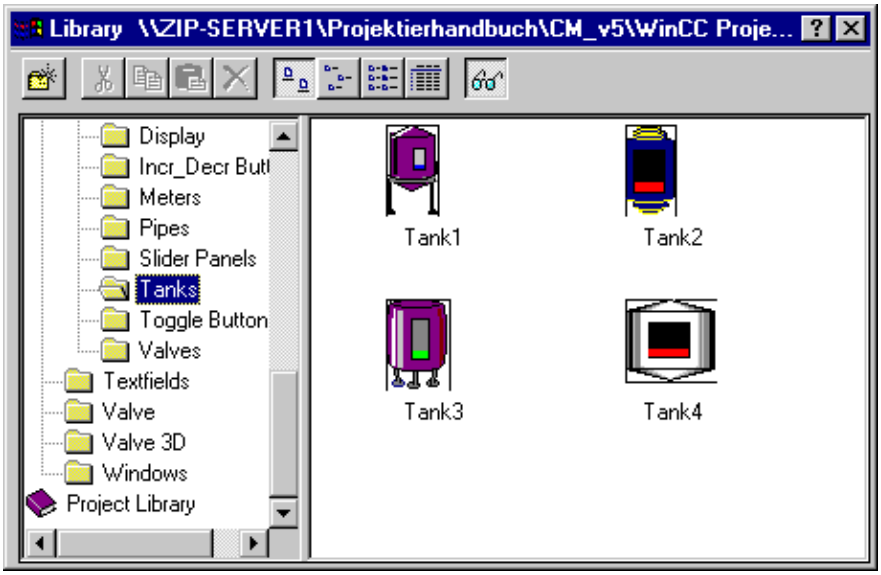
Организовать смену кадров в окне при помощи *кнопки*, управляемой мышью, и прямого соединения.

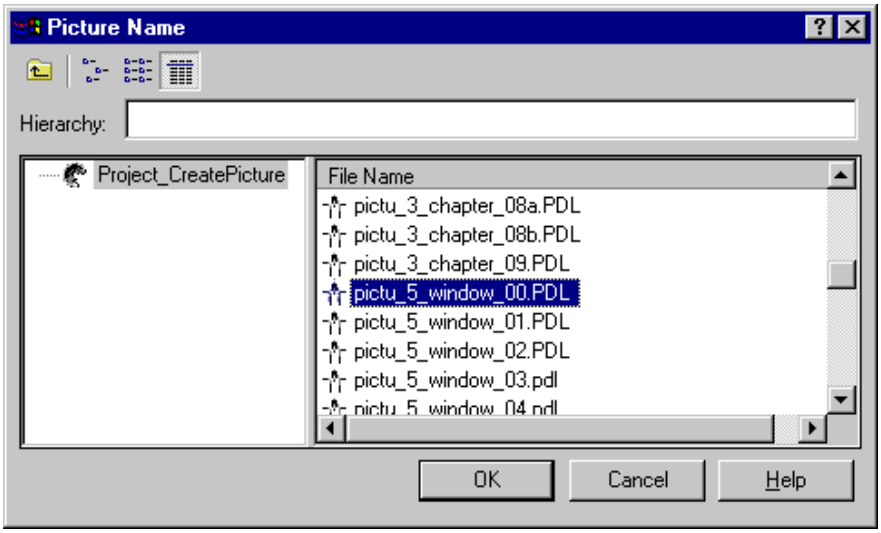
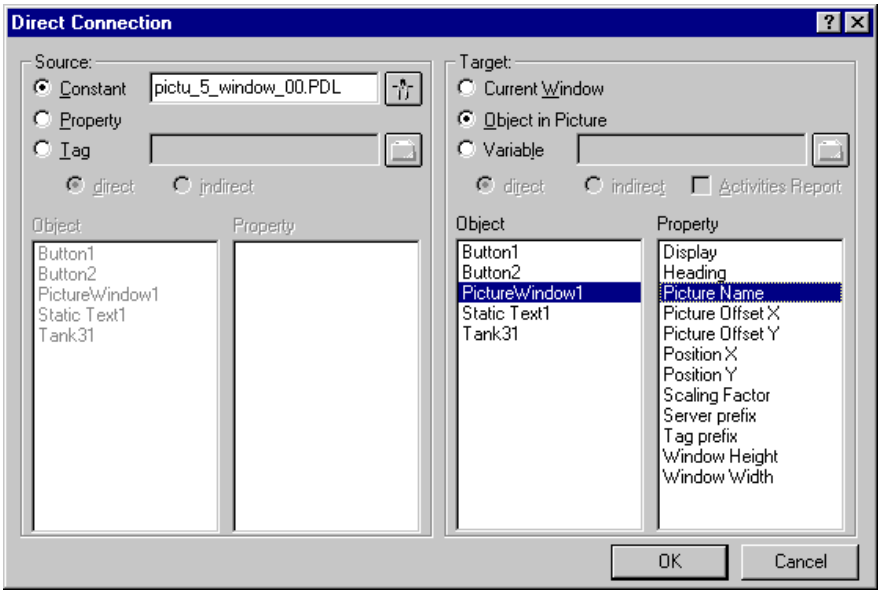
Концепция реализации

При реализации мы будем использовать объект типа *Windows Object (Объект Windows)* → *Button (Кнопка)*, который будет производить смену кадра, отображаемого в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* при щелчке (мыши). Название отображается в кадре при помощи объекта *Standard Object (Стандартный объект)* → *Static Text (Статический текст)*.

Реализация в проекте WinCC

Шаг	Процедура: Открытие кадра с помощью прямого соединения и отображение его названия
1	Командой *File (Файл)* → *New (Новый)* создайте новый кадр и, используя пункт меню *File (Файл)* → *Save As... (Сохранить как...)*, сохраните его под именем *pictu_5_window_00.pdl*. Установите атрибут *Properties (Свойства)* → *Geometry (Геометрия)* → *Width (Ширина)* в *270* и *Properties (Свойства)* → *Geometry (Геометрия)* → *Height (Высота)* в *280*.
2	В кадре *pictu_5_window_00.pdl* создайте *Standard Object (Стандартный объект)* → *Static Text (Статический текст)*. В данном примере используется объект *Static Text1*. Установите атрибут *Properties (Свойства)* → *Font (Шрифт)* → *Bold (Полужирный)* в состояние *Yes (Да)*. В пункте *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* удалите текст по умолчанию из колонки *Static (Статический)*. Это предотвратит появление неверного текста на этапе создания рисунка. Сделайте атрибут *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* динамическим, используя *процедуру Cи*. Эта *процедура Cи* должна возвращать текущее название кадра. В качестве триггера для *процедуры Cи* используется *Default Cycle (Цикл по умолчанию)* → *1 h* (низкий уровень системной загрузки, отсутствие изменений). 

Шаг	**Процедура: Открытие кадра с помощью прямого соединения и отображение его названия**
3	В кадре *pictu_5_window_00.pdl* разместите отображаемые объекты. В примере используется объект *tank3* из глобальной библиотеки. Доступ к библиотеке осуществляется по команде меню *View (Вид)* → *Library (Библиотека)* или по кнопке  на панели инструментов. Убедитесь, что с помощью кнопки  выбран символьный режим обзора для предварительного просмотра отдельный объектов. 
4	Создайте еще два сменных кадра, сохранив созданный кадр командой *File (Файл)* → *Save As... (Сохранить как...)* под именем *pictu_5_window_01.pdl* и еще раз под именем *pictu_5_window_02.pdl*. Это даст нам две копии *pictu_5_window_00.pdl*. После этого во вновь созданных кадрах разместите необходимые визуальные объекты. Изменять объект *Static Text1* для отображения названия кадра не нужно.
5	Создайте новый кадр с помощью команды *File (Файл)* → *New (Новый)*. Поместите в этот кадр *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В данном примере используется объект *Picture Window1*. Установите размеры *окна кадра* в полях *Properties (Свойства)* → *Geometry (Геометрия)* → *Width (Ширина)* и *Properties (Свойства)* → *Geometry (Геометрия)* → *Height (Высота)* в соответствии с размерами ранее созданных кадров. Для отображения рамки во время исполнения установите атрибут *Property (Свойство)* → *Miscellaneous (Разные)* → *Border (Рамка)* в состояние *Yes (Да)*.

Шаг	Процедура: Открытие кадра с помощью прямого соединения и отображение его названия
6	В поле *Properties (Свойства)* → *Miscellaneous (Разные)* → *Picture Name (Название кадра)* укажите *pictu_5_window_00.pdl*. Это заставит программу отображать объект *Picture Window1* при открытии кадра. 
7	В том же кадре создайте объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере это кнопка *Button1*. Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте прямое соединение. Выберите элемент *Constant (Константа)* в качестве *Source (Источника)* и щелкните по активной теперь кнопке  для отображения списка всех доступных кадров. Выберите кадр *pictu_5_window_00.pdl*, *приемник Object in Picture (Объект кадра) Picture Window1* и атрибут *Picture Name (Название кадра)*. Внесите изменения, щелкнув по кнопке *OK*. 

Шаг	Процедура: Открытие кадра с помощью прямого соединения и отображение его названия
8	Используйте (мышь) для выбора сконфигурированного объекта *Button1* и создайте его копию при помощи команды *Edit (Правка)* → *Duplicate (Сделать копию)*. Повторите эту процедуру еще раз. Теперь у нас есть две кнопки, *Button2* и *Button3*. В поле *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* измените настроенное *прямое соединение*. Для *Button2* установите *источник pictu_5_window_01.pdl*, а для *Button3* — *pictu_5_window_02.pdl*.

Процедура Си для Static Text1

```
#include "apdefap.h"
char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty

{
char    *name = lpszPictureName;
char    *pdest;
int     ch = ':';

//check if picture path contains char
pdest = strrchr( lpszPictureName, ch );
//read only picture name without path
if ( pdest == NULL ) return lpszPictureName;
else {
        name = strcpy(name,strrchr(name,':')+1);
        return name;
        }//else
}
```

Объявляются *переменные Си*.

Производится проверка, что *lpszPictureName* содержит только название кадра. Это делается при помощи функции *strrchr*. Эта функция просматривает *lpszPictureName*. Если кадр отображается в *окне кадра*, *lpszPictureName* содержит его название, включая полный путь.

В первом случае в качестве возвращаемой величины используется непосредственно *lpszPictureName*.

Во втором случае из полного пути выделяется только название кадра, которое и используется в качестве возвращаемой величины.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Объект *Static Text1* может быть перенесен в любое другое *окно кадра*. Этот объект также подходит для хранения в библиотеке проекта. В этом случае он легко может быть вставлен в любой кадр перетаскиванием мышью.

Для прямой связи в объекте *Button1*, имя отображаемого рисунка и имя объекта в *окне кадра* должны соответствовать друг другу.

Отображаемые кадры, содержимое кадров и *окна кадров* должны быть сконфигурированы в соответствии с вашими требованиями. Высота и ширина кадра и *окна кадра* должны быть согласованы.

3.2.2 Открытие кадра с помощью динамического мастера (example 02)

Постановка задачи

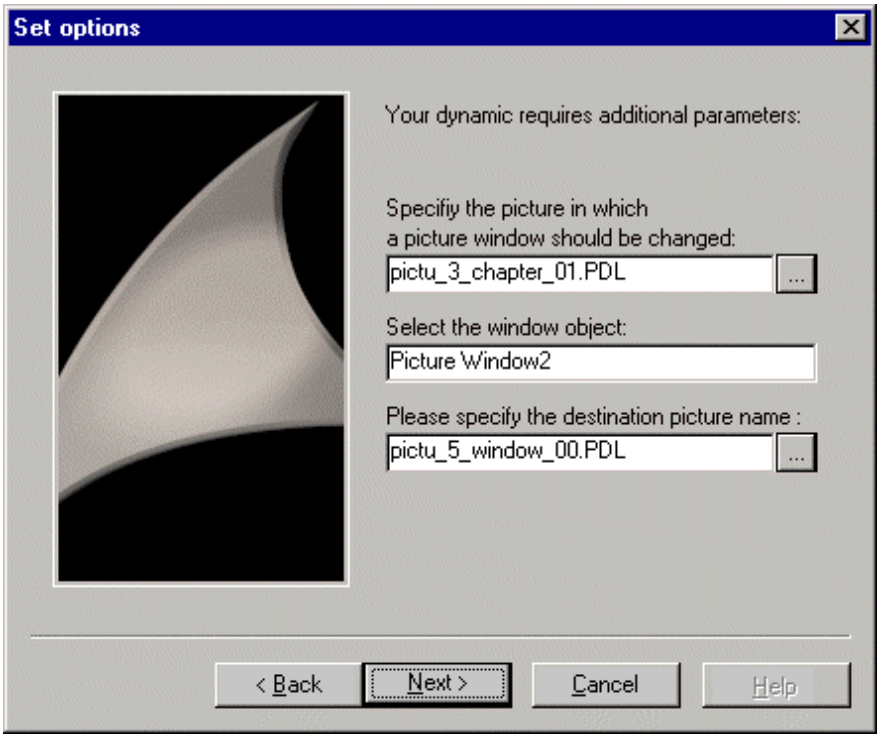
Организовать смену кадров в окне при помощи *кнопки*, управляемой мышью. Реализовать с использованием *Dynamic Wizard (Динамического мастера)*.

Концепция реализации

При реализации мы будем использовать объект типа *Windows Object (Объект Windows)* → *Button (Кнопка)*, который будет производить смену кадра, отображаемого в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* при щелчке на нем  R (правой кнопкой мыши). Мы будем использовать кадры, сконфигурированные в предыдущем примере.

Реализация в проекте WinCC

Шаг	Процедура: Открытие кадра с помощью динамического мастера
1	Поместите в кадр *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В данном примере, это *Picture Window2*. Установите размеры *окна кадра* равными размерам экрана и установите атрибут *Property (Свойство)* → *Miscellaneous (Разные)* → *Border (Рамка)* в состояние *Yes (Да)*. В поле *Properties (Свойство)* → *Miscellaneous (Разные)* → *Picture Name (Название кадра)* выберите кадр *pictu_5_window_01.pdl*.
2	Если *Dynamic Wizard (Динамический мастер)* не отображается, активизируйте его в меню *View (Вид)* → *Toolbars (Панели инструментов)*.

3	В том же кадре создайте объект типа *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется кнопка *Button4*. Выделив объект, выберите пункт *Picture Functions (Функции кадра)* и затем *Picture Change in Window (Смена кадра в окне)* с помощью D (двойного щелчка мыши) в окне *Dynamic Wizard (Динамический мастер)*. На странице *Select Trigger (Выбор триггера) динамического мастера* выберите *Right Mouse Button (Правая кнопка мыши)* и перейдите на следующую страницу, щелкнув на *Next (Следующий)*. Заполните страницу *Set Options (Задание опций)* следующим образом:  С помощью *кнопки* ... можно получить доступ к кадрам проекта. Подтвердите страницу *Finished! (Готово!)*, щелкнув по кнопке *Finish (Завершение)*.
4	Создайте два дополнительных объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. В данном примере это кнопки *Button5* и *Button6*. Аналогичным образом примените к ним *динамический мастер*. На странице *Set Options (Задание опций)* для *Button5* укажите *pictu_5_window_01.pdl* в поле *Destination Picture Name (Название целевого кадра)*, и *pictu_5_window_02.pdl* для *Button6*.

Процедура Си, сгенерированная динамическим мастером

```
#include "apdefap.h"
void OnRButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
static char szPicture[22] = "pictu_5_window_00.PDL";
static char* tmp = &szPicture[0];
PDLRTSetPropEx(PDLRT_AM_PICTABS, "pictu_3_chapter_01",
        "Picture Window2",
        "PictureName",VT_LPSTR,&tmp,NULL,NULL,0,NULL, NULL);
}
```

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Настройки *динамического мастера* должны быть выполнены в соответствии с вашими требованиями.

3.2.3 Открытие кадра с помощью внутренней функции (example 02)

Постановка задачи

Организовать смену кадров в окне при помощи *кнопки*, управляемой мышью. Реализовать с использованием *процедуры Си* для кнопки.

Концепция реализации

Для реализации используем объект типа *Windows Object (Объект Windows)* → *Button (Кнопка)*, который изменяет кадр, отображаемый в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* при нажатии  (мыши). Используем кадры из предыдущего примера.

Концепция проекта WinCC

Шаг	Процедура: Открытие кадра с помощью внутренней функции
1	Создайте в кадре *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В данном примере это *Picture Window2*. Установите размер *окна кадра* в соответствии с размером экрана и атрибут *Property (Свойство)* → *Miscellaneous (Разные)* → *Border* (*Рамка*) в состояние *Yes (Да)*. В поле *Properties (Свойства)* → *Miscellaneous (Разные)* → *Picture Name (Имя кадра)* укажите кадр *pictu_5_window_01.pdl*.
2	В том же кадре создайте объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере использован объект *Button4*. Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте *процедуру Си* для изменения кадра и две дополнительные *кнопки*. В данном случае это объекты *Button5* и *Button6*, снабженные соответственно модифицированными *процедурами Си*.

Процедура Си для Button4

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetPictureName("pictu_3_chapter_01.PDL",
               "Picture Window2",
               "pictu_5_window_00");
}
```

С помощью *внутренней функции SetPictureName (Установка имени кадра)* включите кадр *pictu5_window_00.pdl* в объект *Picture Window2*.
pictu_3_chapter_01.pdl — имя кадра, в котором находится *окно кадра*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Параметры *внутренней функции SetPictureName* должны быть изменены в соответствии с вашими требованиями.

3.2.4 Изменение кадра с помощью динамического мастера (example 03)

Постановка задачи

Организовать смену кадров в окне при помощи *кнопки*, управляемой мышью. Реализовать с использованием *динамического мастера*.

Концепция реализации

При реализации мы будем использовать объект типа *Windows Object (Объект Windows)* → *Button (Кнопка)*, который будет производить смену кадра при щелчке на нем (мышью).

Реализация в проекте WinCC

Шаг	Процедура: Открытие кадра с помощью динамического мастера
1	В данном примере кадр изменяется от *pictu_0_startpicture_00.pdl* к *pictu_3_chapter_01a.pdl*. В примере кадр *pictu_0_startpicture_00.pdl* всегда выбран и, следовательно, смены кадров выполняются только в окнах. Используя *процедуру Си*, сгенерированную *динамическим мастером*, целая система кадров, отображаемая во время выполнения, может быть заменена одним вызываемым кадром. Возврат к *pictu_0_startpicture_00.pdl* сравним с полным перезапуском проекта.
2	Создайте в кадре объект типа *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется объект *Button7*. Picture Change via object name >>
3	При выбранном объекте перейдите на вкладку *Picture Functions (Функции кадра)* и затем к пункту *Single Picture Change (Смена одного кадра)* с помощью D (двойного щелчка мышью) в окне *динамического мастера*. На странице *Select Trigger (Выбор триггера)* выберите пункт списка *Left Mouse Button (Левая кнопка мыши)* и перейдите на следующую страницу, щелкнув по кнопке *Next (Следующий)*. Завершите заполнение страницы *Set Options (Задание опций)* следующим образом: Please specify the new picture name: pictu_3_chapter_03a.PDL ... С помощью *кнопки* ... можно получить доступ к кадрам проекта. Подтвердите страницу *Finished! (Готово!)*, щелкнув кнопку *Finish (Завершение)*.
4	Если смена кадра выполняется в проекте примера, щелкните по кнопке << Back для возврата к проекту.

Процедура Си, созданная динамическим мастером

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF

{
OpenPicture("pictu_3_chapter_01a.PDL");
}
```

Динамический мастер создает процедуру Си. Эта процедура использует *стандартную функцию OpenPicture* для вызова кадра *pictu_3_chapter_01a.pdl* в процессе выполнения. Созданная процедура может также быть перепрограммирована пользователем.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Параметры *динамического мастера* должны быть изменены в соответствии с вашими требованиями.

3.2.5 Изменение одного кадра при помощи прямого соединения (example 04)

Данный пример проекта *Project_CreatePicture* доступен по щелчку на изображенных выше кнопках.

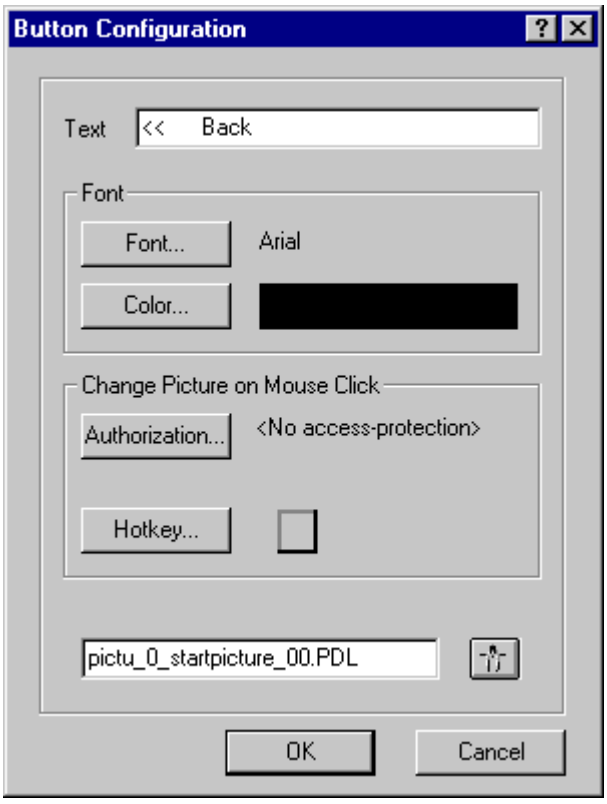
Постановка задачи

В отличие от предыдущих примеров, щелчок на *кнопке*, управляемой мышью, сменит весь кадр. При этом не просто сменится содержимое окна кадра, а будет открыт новый кадр

Концепция реализации

Для реализации используем объект типа *Windows Object (Объект Windows)* → *Button (Кнопка)*, который будет осуществлять смену кадра по щелчку (мыши). Конфигурирование осуществляется с использованием *прямого соединения*.

Реализация в проекте WinCC

Шаг	Процедура: Смена кадра с помощью прямого соединения
1	В данном примере выполняется смена кадра от *pictu_3_chapter_01a.pdl* к *pictu_0_startpicture_00.pdl*.
2	Создайте в кадре объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется объект *Button7*. 
3	В разделе *Change Picture on Mouse Click (Смена кадра по щелчку мыши)* диалога *Button Configuration (Конфигурация кнопки)* при помощи кнопки выбора укажите кадр *pictu_0_startpicture_00*. При этом автоматически будет создано прямое соединение для события *Event (Событие)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)*. Это соединение также может быть создано из диалога *Object Properties (Свойства объекта)*. 

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

В прямом соединении для объекта *Button7* должны быть установлены соответствующие имена кадра и окна кадра.

3.2.6 Открытие кадра по имени объекта с помощью внутренней функции (example 05)

Данный пример проекта *Project_CreatePicture* доступен по щелчку на изображенных выше кнопках.

Постановка задачи

Организовать смену кадров в окне при помощи *кнопки*, управляемой мышью. Кнопка должна определять, какой кадр необходимо открыть по имени своего объекта. Следовательно, кнопку можно использовать повторно только после копирования и смены ее имени.

Концепция реализации

Для реализации используем объект типа *Windows Object (Объект Windows)* → *Button (Кнопка)*, который изменяет кадр, отображаемый в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* при нажатии  (мыши). Используем кадры из предыдущего примера. Имена этих кадров состоят из двух частей — текстовой строки и номера кадра.

Реализация в проекте WinCC

Шаг	Процедура: Открытие кадра по имени объекта с помощью внутренней функцию
1	Создайте в кадре *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В данном примере используется объект *Picture Window1*. Установите размеры о*кна кадра* равными размерам ранее созданных кадров. Для того чтобы во время выполнения окно отображалось в рамке, установите атрибут *Property (Свойство)* → *Miscellaneous (Разные)* → *Border (Рамка)* в состояние *Yes (Да)*. В поле *Properties (Свойства)* → *Miscellaneous (Разные)* → *Picture Name (Имя кадра)* укажите кадр *pictu_5_window_00.pdl*.
2	Создайте в том же кадре объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется объект *Button0*. Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте *процедуру Cи*, которая считывает имя и номер кнопки и отображает необходимый кадр, в соответствии с принятыми соглашениями о формировании имени.
3	Создайте две копии объекта *Button0* и смените их имена на *Button1* и *Button2*.

Процедура Си для Button0

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
char    name[30];
int     number;
int     ch = 'n';
char    *pdest;

//check if object name contains character
pdest = strrchr( lpszObjectName, ch );
if ( pdest == NULL )(printf("ObjectNameError"));
else {
   //read object number
   number = atoi(strrchr(lpszObjectName,'n')+1);
   //generate picture name
   sprintf(name,"pictu_5_window_%02d.PDL",number);
   //set picture name
   SetPictureName("pictu_3_chapter_01a.PDL","Picture Window1",name);
   }
}
```

Объявляются *переменные Си*.

Производится проверка, что имя объекта соответствует принятым соглашениям. Объектам присваиваются имена вида [button]+[номер вызываемого кадра].

Выводится сообщение об ошибке, если символ перед номером, а именно 'n', не найден.

Считывается номер, содержащийся в имени кнопки. Функция *strrchr* сканирует имя в обратном направлении в поисках символа n. Символьная строка, следующая за *n*, преобразуется к целому числу с помощью функции *atoi*.

Функция *sprintf* использует имя и номер кадра для получения полного имени кадра, вызываемого кнопкой.

С помощью *внутренней функции SetPictureName* вызываемый кадр включается в объект *Picture Window1*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Процедура Си для *кнопки* и способ именования объектов должны быть приведены в соответствие с принятыми соглашениями о структуре имен. Убедитесь, что эти соглашения строго соблюдаются как для имен кадров, так и для имен объектов, чтобы гарантировать отсутствие проблем при доступе к нужным кадрам.

3.2.7 Открытие кадра и отображение его названия по имени объекта с помощью соединения с тегом (example 06)

Данный пример проекта *Project_CreatePicture* доступен по щелчку на изображенных выше кнопках.

Постановка задачи

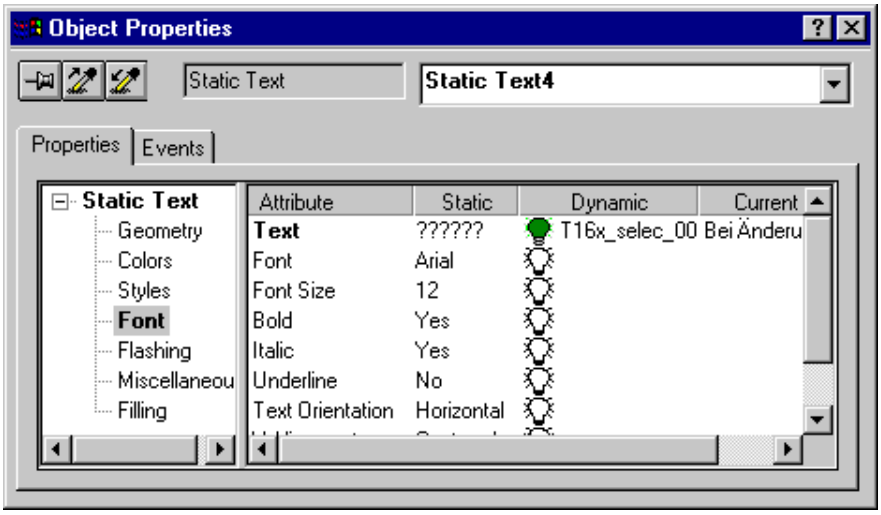
Организовать смену кадров в окне при помощи *кнопки*, управляемой мышью. Кнопка должна определять, какой кадр необходимо открыть по имени своего объекта. Следовательно, кнопку можно использовать повторно только после копирования и смены ее имени. Имя кадра сохраняется в текстовом теге и отображается в текстовом поле, отсутствующем в реальном кадре.

Концепция реализации

Для реализации используем объект типа *Windows Object (Объект Windows)* → *Button (Кнопка)*, который изменяет кадр, отображаемый в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* при нажатии (мыши). Используем кадры из предыдущего примера. Имена этих кадров состоят из двух частей — текстовой строки и номера кадра. Дополнительно будем использовать *Standard Object (Стандартный объект)* → *Static Text (Статический текст)* для отображения имени кадра.

Реализация в проекте WinCC

Шаг	Процедура: Открытие кадра и отображение его названия по имени объекта с помощью соединения с тегом
1	В менеджере тегов создайте тег типа *Text Tag 16–Bit Character Set (Текстовый тег 16–битного набора символов)*. В данном примере используется тег *T16x_selec_00*. Этот тег содержит имя кадра, отображаемого в окне
2	Откройте диалог свойств объекта кадра *pic_chapter_01a.pdl*. Для события *Events (События)* → *Miscellaneous (Разное)* → *Open Picture (Открытие кадра)* создайте *процедуру Cи*, которая назначает имя кадра *pictu_5_window_01.pdl* тегу *T16x_selec_00*. Это соответствует кадру, который должен отображаться при первом открытии.
3	В кадре создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В данном примере используется *Picture Window2*. Установите размеры *окна кадра* равными размерам ранее созданных кадров. Для того чтобы во время выполнения окно отображалось в рамке, установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Border (Рамка)* в состояние *Yes (Да)*. В поле *Properties (Свойства)* → *Miscellaneous (Разные)* → *Picture Name (Имя кадра)* укажите *pictu_5_window_01.pdl* и установите связь с тегом *T16x_selec_00*.

4	В том же кадре создайте объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется объект *Button_0*. Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте *процедуру Си*, которая считывает имя и номер кнопки и присваивает это имя внутреннему тегу *T16x_selec_00*.
5	Продублируйте объект *Button_0* два раза и смените имена полученных объектов на *Button_1* и *Button_2*, соответственно.
6	Создайте в кадре *Smart Object (Интеллектуальный объект)* → *Static Text (Статический текст)* над окном кадра *Picture Window2*. В данном примере используется объект *Static Text4*. Установите атрибут *Properties (Свойства)* → *Font (Шрифт)* → *Bold (Полужирный)* в состояние *Yes (Да)*. В поле *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* удалите введенный текст из колонки *Static (Статический)* и создайте соединение с тегом *T16x_selec_00 tag*. Установите режим обновления *Upon Change (По изменению)*. Уничтожение статического элемента предотвращает появление неверного текста в момент вывода кадра. 

Процедура Си для открытия кадра

```
#include "apdefap.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpszPi

{
SetTagChar("T16x_selec_00","pic_window_01.pdl");
}
```

Имя кадра формируется при помощи *внутренней функции SetTagChar*.

Процедура Си для Button_0

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
char name[30];
int number;
int  ch = '_';
char *pdest;

//check if object name contains character
pdest = strrchr( lpszObjectName, ch );
if ( pdest == NULL )(printf("ObjectNameErrcr"));
else {
   //read object number
   number = atoi(strrchr(lpszObjectName,'_')+1);
   //generate picture name
   sprintf(name,"pictu_5_window_%02d.PDL",number);
   //set tag which contains picture name
   SetTagChar("T16x_selec_00",name);
   }
}
```

Объявляются *переменные Си*.

Производится проверка, что имя объекта соответствует принятым соглашениям. Объектам присваиваются имена вида [button]+[номер вызываемого кадра].

Выводится сообщение об ошибке, если символ перед номером, а именно 'n', не найден.

Считывается номер, содержащийся в имени кнопки. Функция *strrchr* сканирует имя в обратном направлении в поисках символа n. Символьная строка, следующая за *n*, преобразуется к целому числу с помощью функции *atoi*.

Функция *sprintf* использует имя и номер кадра для получения полного имени кадра, вызываемого кнопкой.

С помощью *внутренней функции SetTagChar* имя вызываемого кадра записывается в тег *T16x_selec_00*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Процедура Си для *кнопки* и способ именования объектов должны быть приведены в соответствие с принятыми соглашениями о структуре имен. Убедитесь, что эти соглашения строго соблюдаются как для имен кадров, так и для имен объектов, чтобы гарантировать отсутствие проблем при доступе к нужным кадрам.

3.3 Отображение окна кадра

Примеры, имеющие отношение к этой теме, доступны в проекте *Project_CreatePicture* по нажатию (мышью) на *кнопке*, изображенной выше. Примеры приведены в кадре *pictu_3_chapter_03.pdl*.

3.3.1 Скрытие (отмена выбора) и отображение (выбор) извне окна кадра (example 01)

Постановка задачи

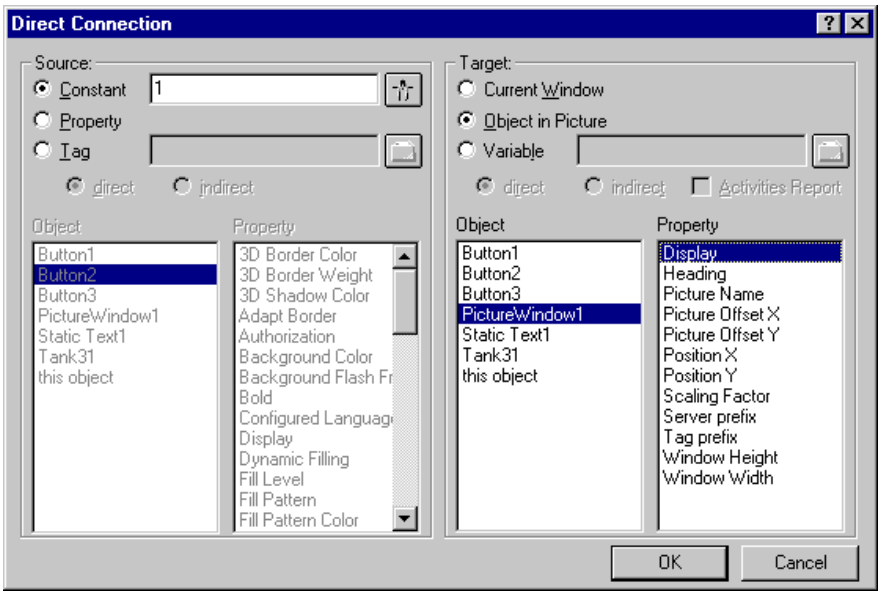
Организовать динамическое отображение и скрытие *окна кадра* с использованием двух кнопок, управляемых мышью.

Концепция реализации

Для реализации будем использовать два объекта типа *Windows Objects (Объекты Windows)* → *Buttons (Кнопка)*, которые будут отображать и скрывать кадр в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* при щелчке (мыши).

Реализация в проекте WinCC

Шаг	Процедура: Скрытие и отображение извне окна кадра
1	Создайте кадр, который будет отображаться и скрываться, т.е. вспомогательный текст или информационное сообщение. В данном примере используется *pictu_5_window_07* — информационная панель без дополнительных элементов управления.
2	В другом кадре создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно Кадра)* того же размера, что и ранее созданный кадр. В данном примере используется объект *Picture Window1*. Установите атрибут *Property (Свойство)* → *Geometry (Геометрия)* → *Width (Ширина)* в *246* и *Property (Свойство)* → *Geometry (Геометрия)* → *Height (Высота)* в *129*. Для того чтобы во время выполнения окно отображалось с рамкой, установите атрибут *Property (Свойство)* → *Miscellaneous (Разное)* → *Border (Граница)* в состояние *Yes (Да)*. Для разрешения перемещения окна установите атрибут *Property (Свойство)* → *Miscellaneous (Разное)* → *Moveable (Перемещаемый)* в состояние *Yes (Да)*. Для скрытия окна во время выполнения установите атрибут *Property (Свойство)* → *Miscellaneous (Разное)* → *Display (Отображение)* в состояние *No (Нет)*. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* укажите кадр *pictu_5_window_07.pdl*

Шаг	Процедура: Скрытие и отображение извне окна кадра
3	В том же кадре создайте два дополнительных объекта *Windows Objects (Объекты Окна)* → *Buttons (Кнопки)*. В данном примере это объекты *Button1* и *Button2*. У *Button1* создайте прямое соединение для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Соедините источник *Constant (Постоянная)* → *1* с приемником *Object in Picture (Объект кадра)* → *Picture Window1* → *Display (Отображение)*. Внесите изменения, щелкнув по кнопке *OK*. 
4	Таким же образом, как для *Button1*, создайте прямое соединение у *Button2* для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. В качестве константы укажите число 0.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Для *прямых соединений* объектов *Button1* и *Button2* необходимо скорректировать отображаемое имя кадра и имя *окна кадра*.

Прилагаемый кадр *pictu_5_window_07* можно просто перенести в другой проект, предварительно изменив заголовок и информационный текст.

3.3.2 Отображение (выбор) извне и скрытие (отмена выбора) из окна кадра (example 02)

Постановка задачи

Динамически отображать *окно кадра* по щелчку мышью на кнопке. Динамически скрывать окно кадра по щелчку на кнопке в *окне кадра*.

Концепция реализации

Для реализации используем два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*, которые будут отображать и скрывать кадр в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно Кадра)* при щелчке  (мыши).

Реализация в проекте WinCC

Шаг	Процедура: Отображение (выбор) извне и скрытие (отмена выбора) из окна кадра
1	Создайте кадр, который должен отображаться и скрываться, т.е. вспомогательный текст или информационное сообщение. В данном примере используется кадр *pictu_5_window_08* — информационная панель с дополнительным объектом *Windows Object (Объект Windows)* → *Button (Кнопка)*, с помощью которой производится отмена выбора кадра. В этом примере используется объект *Button1*.
2	У *Button1* создайте *прямое соединение* для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой клавиши)*. Свяжите источник *Constant (Константа)* → *0* с приемником *Current Window (Текущее окно)* → *Display (Отображение)*. Внесите изменения, щелкнув по кнопке *OK*.
3	В другом кадре создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* того же размера, что и ранее созданный кадр. В данном примере используется объект *Picture Window2*. Установите *Property (Свойство)* → *Geometry (Геометрия)* → *Width (Ширина)* в *246* и *Property (Свойство)* → *Geometry (Геометрия)* → *Height (Высота)* в *129*. Для отображения окна с рамкой во время выполнения установите *Property (Свойство)* → *Miscellaneous (Разные)* → *Border (Рамка)* в состояние *Yes (Да)*. Для того чтобы окно можно было перемещать, установите *Property (Свойство)* → *Miscellaneous (Разные)* → *Moveable (Перемещаемый)* в состояние *Yes (Да)*. Для скрытия окна во время выполнения установите *Property (Свойство)* → *Miscellaneous (Разные)* → *Display (Отображение)* в состояние *No (Нет)*. В поле *Properties (Свойства)* → *Miscellaneous (Разные)* → *Picture Name (Имя кадра)* укажите кадр *pictu_5_window_08.pdl*.

4	В том же кадре создайте объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере это объект *Button3*. У *Button3* создайте прямое соединение для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Соедините источник *Constant (Константа)* → *1* с приемником *Object in Picture (Объект кадра)* → *Picture Window2* → *Display (Отображение)*. Внесите изменения, щелкнув по кнопке *OK*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Для *прямого соединения* объекта *Button3* необходимо скорректировать отображаемое имя кадра и имя *окна кадра*.

Прилагаемый кадр *pictu_5_window_08* можно просто перенести в другой проект, предварительно изменив заголовок и информационный текст. *Прямое соединение* для *Button1* модифицировать не нужно.

3.3.3 Скрытие кадра по времени (example 03)

Постановка задачи

Организовать динамическое отображение и скрытие *окна кадра* с использованием двух кнопок, управляемых мышью. По истечении определенного времени окно кадра должно автоматически скрываться.

Концепция реализации

Для реализации будем использовать объект *Windows Object (Объект Windows)* → *Button (Кнопка)*, который отображает и прячет *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* при нажатии кнопки  (мыши).

Реализация в проекте WinCC

Шаг	Процедура: Скрытие кадра по времени
1	Создайте кадр, который должен отображаться и скрываться, то есть вспомогательный текст или информационное сообщение. В данном примере использован кадр *pictu_5_window_09* — информационная панель без дополнительных элементов управления. Для реализации скрытия по времени объекта *Graphic Object1* для атрибута *Properties (Свойства)* → *Geometry (Геометрия)* → *Position X (Координата X)* создается *процедура Cu*. Она может быть размещена где угодно, так как требуется только один *триггер*. Установите в качестве триггера *1 s*. 
2	В другом кадре создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* того же размера, что и ранее созданный кадр. В данном примере используется объект *Picture Window3*. Установите *Property (Свойство)* → *Geometry (Геометрия)* → *Width (Ширина)* в *246* и *Property (Свойство)* → *Geometry (Геометрия)* → *Height (Высота)* в *129*. Для отображения окна с рамкой во время выполнения установите *Property (Свойство)* → *Miscellaneous (Разные)* → *Border (Рамка)* в состояние *Yes (Да)*. Для того чтобы окно можно было перемещать, установите *Property (Свойство)* → *Miscellaneous (Разные)* → *Moveable (Перемещаемый)* в состояние *Yes (Да)*. Для скрытия окна во время выполнения установите *Property (Свойство)* → *Miscellaneous (Разные)* → *Display (Отображение)* в состояние *No (Нет)*. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* укажите кадр *pictu_5_window_09.pdl*.
3	Создайте кнопку, в этом примере используется объект *Button4*. Создайте *процедуру Cu* у *Button4* для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*, которая отображает и прячет окно кадра.

Процедура Си для Graphic Object1

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty

{
static int i = 0;

//count time
i++;
//if maximum time is reached
if (i>5) SetVisible("pictu_3_chapter_03.PDL","Picture Window3",0);
return 0;
}
```

Объявляется статическая *переменная Си*. Эта переменная сохраняет свое значение в течение всего времени, пока открыт кадр.

При каждом вызове процедуры инкрементируется статическая *переменная Си*.

Если *i* превышает 5, т.е., для односекундного триггера через 5 с., окно кадра скрывается.

Возвращаемая величина — координата X *Graphic Object1*, так как *процедура Си* создана для этого атрибута, но само значение атрибута не меняется.

Процедура Си для Button4

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lps
{
//set visibility in complement state
SetVisible(lpszPictureName,"Picture Window3",
        !(SHORT)!GetVisible(lpszPictureName,"Picture Window3"));
}
```

Признак видимости *окна кадра Picture Window3* изменяется на противоположный с помощью внутренней функции *SetVisible*. Текущее состояние запрашивается внутренней функцией *GetVisible*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Для *процедуры Си* объекта *Button4* необходимо скорректировать отображаемое имя кадра и имя *окна кадра*.

Прилагаемый кадр *pictu_5_window_09* можно просто перенести в другой проект, предварительно изменив заголовок и информационный текст. В *процедуре Си* для *Graphic Object1* время до скрытия кадра может определяться пользователем путем изменения триггера или условия в конструкции условного перехода.

3.3.4 Отображение окна кадра при нажатии правой кнопки мыши (example 04)

Постановка задачи

Отображать окно кадра должно при нажатии *кнопки* R (правой кнопкой мыши) и скрывать при отпускании R (правой кнопки мыши).

Концепция реализации

Для реализации используем *Windows Object (Объект Windows)* → *Button (Кнопка)*, с помощью которой *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* делается видимым во время нажатия R (правой кнопки мыши).

Реализация в проекте WinCC

Шаг	Процедура: Отображение окна кадра при нажатии правой кнопки мыши
1	Создайте кадр, который должен отображаться и скрываться, т.е. текст подсказки или информационную панель. В примере используется кадр *pictu_5_window_07* — информационная панель без дополнительных элементов управления.
2	В другом кадре создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* тех же размеров, что и ранее созданный кадр. В данном примере используется объект *Picture Window1*. Установите *Property (Свойство)* → *Geometry (Геометрия)* → *Width (Ширина) — 246* и *Property (Свойство)* → *Geometry (Геометрия)* → *Height (Высота) — 129*. Для отображения окна с рамкой во время исполнения, установите *Property* → *Miscellaneous (Разное)* → *Border (Рамка)* в *Yes (Да)*. Для того чтобы окно можно было перемещать, установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Moveable (Перемещаемое)* в *Yes (Да)*. В поле *Properties (Свойство)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* установите *pictu_5_window_07.pdl*.
3	В том же кадре создайте *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется объект *Button5*. Для *Button5* создайте *прямое соединение* с *Events (События)* → *Mouse (Мышь)* → *Press Right (Нажатие правой кнопки)*. Соедините *источник Constant (Константа)* → *1* с *приемником Object in Picture (Объект кадра)* → *Picture Window4 (Окно кадра 4)* → *Display (Отображение)*. Подтвердите установки щелчком по кнопке *OK*.
4	Аналогичным образом создайте *прямое соединение* с *Events (События)* → *Mouse (Мышь)* → *Release Right (Отпускание правой кнопки)*. В качестве *константы* укажите значение *0*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Необходимо скорректировать отображаемое имя кадра и имя *окна кадра* для *прямого соединения* в *Button5*.

Прилагаемый кадр *pictu_5_window_07* можно непосредственно перенести в другой проект после модификации заголовка и информационного текста.

3.3.5 Создание информационных панелей с помощью мастера (example 05)

Пример доступен при выборе изображенной выше кнопки с помощью  (мыши).
Пример приведен в кадре *pictu_3_chapter_03a.pdl*.

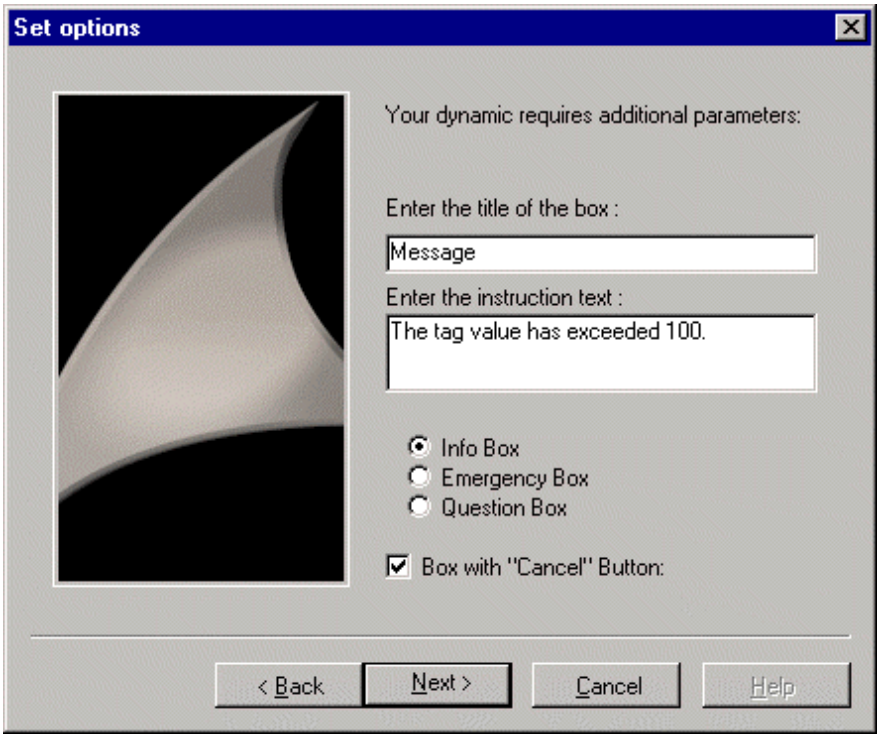
Постановка задачи

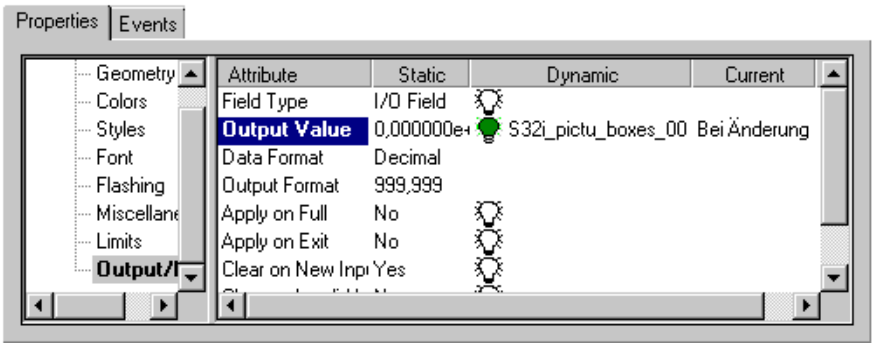
Отображать информационную панель, если значение тега превышает 100, и аварийную панель, если значение тега превышает 150.

Концепция реализации

Для реализации используем *Windows Object (Объект Windows)* → *Slider Object (Бегунок)* для ввода тега и *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)* для отображения значения тега.

Реализация в проекте WinCC

Шаг	Процедура: Создание информационных панелей с помощью мастера
1	Если *динамический мастер* не отображается, активируйте его командой *View (Вид)* → *Toolbars (Панели инструментов)*.
2	Создайте в кадре *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода–вывода)*. В данном примере используется объект *I/O Field1*. Выделив объект, выберите вкладку *Picture Functions (Функции кадра)* и затем *Display Instruction Box (Отображение панели команд)* с помощью D (двойного щелчка мыши) из динамического мастера. На странице динамического мастера *Select Trigger (Выбор триггера)*, выберите *Left Mouse Button list (Список функций левой клавиши)* и перейдите на следующую страницу, щелкнув по кнопке *Next (Далее)*. Заполните страницу *Set Options (Установка опций)* следующим образом:  Подтвердите страницу *Finished! (Готово!)*, щелкнув по кнопке *Finish* (Завершить).
3	Вновь используйте динамический мастер для *I/O Field1*. На странице выбора триггера, выберите *Right Mouse Button (Правая кнопка мыши)*; на странице *Set Options*, выберите радио–кнопку *Emergency Box (Аварийная панель)* и введите текст сообщения.
4	В *менеджере тегов* создайте тег типа *Signed 32–Bit Value (32–битная величина со знаком)*. В данном примере используется тег *S32i_pictu_boxes_00*.

Шаг	Процедура: Создание информационных панелей с помощью мастера
5	В том же кадре создайте *Windows Object (Объект Windows)* → *Slider Object (Бегунок)*. В данном примере это *Slider Object1*. Для него создайте *прямое соединение* с *Events (События)* → *Property Topics (Разделы свойств)* → *Miscellaneous (Разное)* → *Process Driver Connection (Соединение с драйвером процесса)*. Соедините *источник Property (Свойство)* → *Slider Object1* → *Process Driver Connection (Соединение с драйвером процесса)* с *приемником Variable (Переменная)* → *S32i_pictu_boxes_00*. Подтвердите установки, щелкнув по кнопке *OK*.
6	У объекта *I/O Field1* для *Properties (Свойства)* → *Output/Input (Вывод/Ввод)* → *Output Value (Выводимое значение)* создайте *соединение с тегом S32i_pictu_boxes_00* и триггер по изменению. 
7	У объекта *I/O Field1* создайте *процедуру Cи* для *Events (События)* → *Property Topics (Разделы свойств)* → *Output/Input (Вывод/ввод)* → *Output Value(Выводимое значение)*, которая отображает информационную панель, если тег *S32i_pictu_boxes_00* превышает значение 100, и аварийную панель при превышении 150. Фрагменты *процедур Cи*, созданных *динамическим мастером* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* и *Press Right (Нажатие правой кнопки)* могут быть скопированы и вставлены в эту *процедуру Cи*.
8	Уничтожьте *процедуру Cи* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* и *Press Right (Нажатие правой кнопки)*.

Процедура Си для I/O Field1

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l

{
int a;
static int i = 0, j = 0;

//get tag value
a=GetTagDWord("S32i_pictu_boxes_00");

//set visible info box
if ((a>100)&&(i==0)) {
        i=1;
        MessageBox(NULL,"Der Variablenwert hat\r\n100 überschritten",
        "Hinweis", MB_OK|MB_ICONEXCLAMATION|MB_SETFOREGROUND);
        }//if
if (a<=100) (i=0);

//set visible emergency box
if ((a>150)&&(j==0)) {
        j=1;
        MessageBox(NULL,"Der Variablenwert hat\r\n150 überschritten",
        "Achtung!!!", MB_OK|MB_ICONSTOP|MB_SETFOREGROUND);
        }//if
if (a<=150) (j=0);
}
```

Считайте значение тега, используя *внутреннюю функцию GetTagDWord*.

Если значение превышает 100, при помощи *процедуры Си*, созданной *динамическим мастером*, отображается информационная панель. После превышения значения 100, информационная панель будет закрыта только после того, как значение станет меньше 100, т.е. после обнуления статической переменной Си.

Если значение превышает 150, при помощи *процедуры Си*, созданной *динамическим мастером*, отображается аварийная панель. После превышения значения 150, аварийная панель будет закрыта только после того, как значение станет меньше 150, т.е. после обнуления статической переменной Си.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Для *процедуры Си* в поле *I/O Field1* имя переменной должно быть скорректировано.

Текст, отображаемый в информационной и аварийной панелях, должен быть приведен в соответствие с вашими требованиями.

3.3.6 Отображение диалога для ввода текста (example 06)

Пример доступен при выборе изображенной выше кнопки с помощью (мыши) .
Пример приведен в кадре *pictu_3_chapter_03a.pdl*.

Постановка задачи

Отображать диалог ввода текста при нажатии *кнопки* (мышью). Отображать вводимый текст в кадре.

Концепция реализации

В реализации используем *Windows Object (Объект Windows)* → *Button (Кнопка)* для открытия диалога, и *Standard Object (Стандартный объект)* → *Static Text (Статический текст)* для отображения текста. Для ввода текста в диалоге используем *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)* и два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)* для подтверждения или отказа от введенного текста.

Реализация в проекте WinCC

Шаг	Процедура: Отображение диалога для ввода текста
1	В *менеджере тегов* создайте два тега типа *Text Tag 16–Bit Character Set (Текстовый тег 16–битного набора символов)*. В данном примере используются теги *T16i_pictu_win_00* и *T16i_pictu_win_01*.
2	Создайте кадр, в котором должен выполняться ввод текста. В примере использован кадр *pictu_5_window_17.pdl*.
3	Создайте в этом кадре *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В диалоге при его создании выберите тег *T16i_pictu_win_01* и установите триггер *Upon Change (По изменению)*. Установите *Property (Свойство)* → *Output/Input (Вывод/ввод)* → *Data Format (Формат данных) — String (Строка)* и *(Свойство)* → *Output/Input (Вывод/ввод)* → *Apply on Exit (Применить при выходе) – Yes (Да)*. Это делает подтверждение введенного текста клавишей ENTER необязательным.
4	Создайте в том же кадре объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется *Button1*. Эта кнопка используется для приема введенного текста. Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте *прямое соединение* с *источником Variable T16i_pictu_win_01* и *приемником Variable T16i_pictu_win_00*. Для события *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* создайте *прямое соединение*, которое скрывает кадр.
5	Создайте другой объект *Windows Object (Объект Windows)* → *Button*

Шаг	Процедура: Отображение диалога для ввода текста
	(Кнопка). В данном примере это объект *Button2*. Эта кнопка используется для отказа от ввода и сохранения введенного ранее текста. Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте *прямое соединение* с *источником Variable T16i_pictu_win_00* и *приемником Variable T16i_pictu_win_01*. Это прямое соединение передает содержимое *T16i_pictu_win_00* (включая старый текст) в *T16i_pictu_win_01*. Для события *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* создайте *прямое соединение*, которое скрывает кадр.
6	Создайте во втором кадре *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В данном примере используется объект *Picture Window1*. Установите размеры *окна кадра* в соответствии с созданным кадром. Если *окно кадра* должно отображаться с рамкой, *высота* и *ширина окна кадра* должны быть на 10 пикселей больше, чем те же величины самого кадра. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* введите *pictu_5_window_17.pdl*.
7	Создайте в том же кадре объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере это объект *Button1*. Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте прямое соединение. Соедините *источник Constant (Константа)* → *1* с *приемником Object in Picture (Объект кадра)* → *Picture Window1* → *Display (Отображение)*. Подтвердите изменения щелчком по кнопке *OK*.
8	Создайте в том же кадре *Standard Object (Стандартный объект)* → *Static Text (Статический текст)*. В данном примере это объект *Static Text1*. В поле *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* создайте *соединение с тегом T16i_pictu_win_00* и установите триггер по изменению.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Кадр *pictu_5_window_17.pdl* может использоваться для ввода текста, однако *процедуры Си* для *кнопок* должны быть приведены в соответствие с вашими именами переменных.

3.4 Разрешение управления оператором

К примерам, относящиеся к этой теме, можно получить доступ в проекте *Project_CreatePicture*, выбрав изображенную выше *кнопку* с помощью (мыши). Примеры приведены в кадре *pictu_3_chapter_02.pdl*.

3.4.1 Выход из режима исполнения и системы (example 01)

Постановка задачи

Использовать две управляемые мышью кнопки для выбора диалоговых окон, позволяющих выйти из режима исполнения или из системы в целом.

Концепция реализации

Для реализации используем два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*, каждый из которых отображает кадр в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, при щелчке (мыши). В отдельных кадрах два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)* позволяют либо вызвать соответствующую системную функцию, либо отменить действие.

Реализация в проекте WinCC

Шаг	Процедура: Выход из режима исполнения и системы
1	Создайте кадр, который собираетесь использовать для выхода из режима исполнения. В примере используется кадр *pictu_5_window_04.pdl*.
2	Создайте в этом кадре объект *Windows Object (Объект Windows)* → *Button (Кнопка)* в примере используется объект *Button1*. При выделенном объекте выберите вкладку *System Functions (Системные функции)* и затем пункт *Exit WinCC or Windows (Выход из WinCC или Windows)* с помощью D (двойного щелчка) на *динамическом мастере*. На странице *Выбор триггера динамического мастера* выберите пункт *Left Mouse Button (Левая кнопка мыши)* и перейдите на следующую страницу, щелкнув (мышью) по кнопке *Next (Далее)*. На странице *Set Options (Установка опций)* выберите *Exit Windows (Выход из Windows)*. Подтвердите страницу *Finished! (Готово!)*, щелкнув по кнопке *Finish (Завершение)*.
3	Создайте другой объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется объект *Button2*. Эта кнопка используется для завершения процедуры. Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте *прямое соединение*, которое будет скрывать кадр.
4	Создайте другой кадр, который собираетесь использовать для выхода из системы. В примере используется кадр *pictu_5_window_03.pdl*.
5	Создайте в этом кадре объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется объект *Button1*. При выделенном объекте выберите вкладку *System Functions (Системные функции)* и затем пункт *Exit WinCC Runtime (Выход из режима исполнения WinCC)* с помощью D (двойного щелчка мыши) на *динамическом мастере*. На странице *динамического мастера Выбор триггера* выберите пункт *Left Mouse Button (Левая кнопка мыши)* и перейдите на следующую страницу, щелкнув по кнопке *Next (Далее)*. Подтвердите страницу *Finished! (Готово!)*, щелкнув по кнопке *Finish (Завершение)*.

Шаг	Процедура: Выход из режима исполнения и системы
6	Создайте другой объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется объект *Button2*. Эта кнопка используется для отмены действия. Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте *прямое соединение*, которое будет скрывать кадр.
7	Создайте в другом кадре два *Smart Objects (Интеллектуальные объекты)* → *Picture Windows (Окна кадра)*. В данном примере используются объекты *Picture Window1* и *Picture Window2*, которые расположены одно над другим. Установите размеры *окон кадра* в соответствии с размерами созданных кадров. Если *окна кадра* должны отображаться с рамками, *высота* и *ширина окон кадра* должны быть установлены на 10 пикселей больше, чем соответствующие величины кадров, для того чтобы отобразить весь кадр. В полях *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* введите соответствующие названия кадров. Установите атрибут *Property (Свойство)* → *Miscellaneous (Разное)* → *Display (Отображение)* в состояние *No (Нет)*.
8	Создайте в том же кадре два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. В данном примере это объекты *Button1* и *Button2*. У *Button1* создайте *прямое соединение* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Соедините *источник Constant (Константа)* → *1* с *приемником Object in Picture (Объект кадра)* → *Picture Window1* → *Display (Отображение)*. Подтвердите установки, щелкнув по кнопке *OK*. Аналогичным образом создайте *прямое соединение* для *Button2*, но установите в качестве *приемника Object in Picture (Объект кадра)* → *Picture Window2* → *Display (Отображение)*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Кадры для выхода из системы и режима исполнения могут непосредственно использоваться в других проектах.

У *кнопок* для вызова *окон кадра* должны быть скорректированы названия объектов в прямом соединении.

3.4.2 Разрешение управления оператором, стандартная панель входа в систему (example 02)

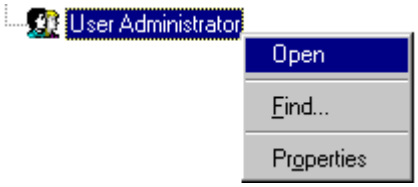
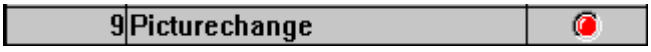
Постановка задачи

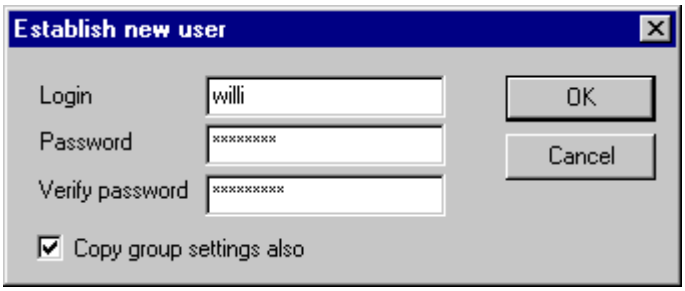
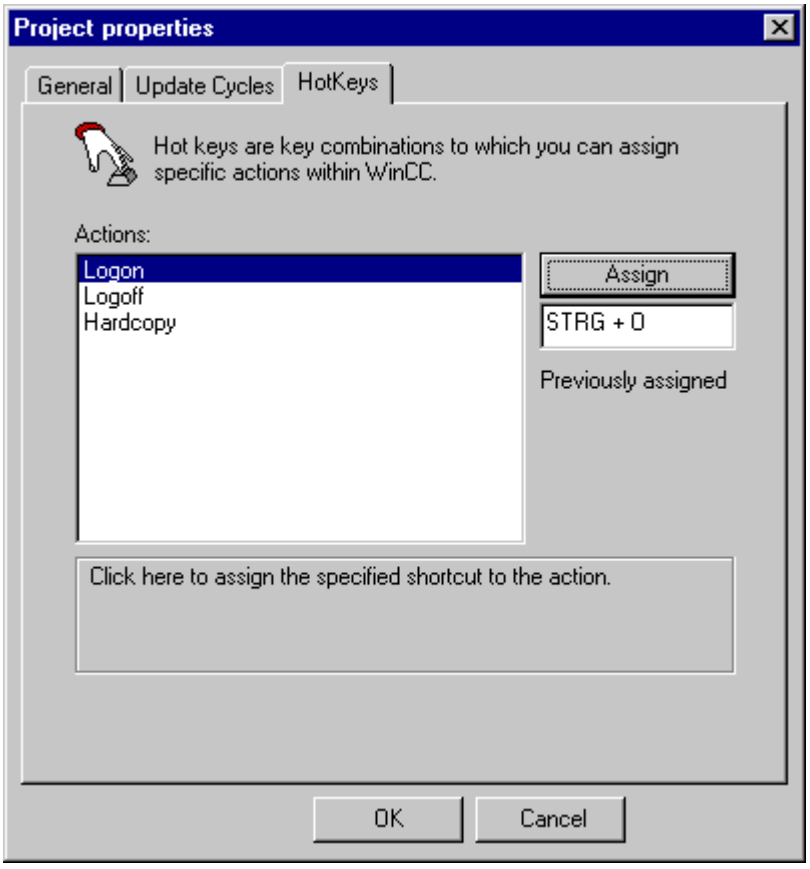
С помощью двух *кнопок* производить изменение кадра , только в том случае, если пользователь имеет соответствующую авторизацию.

Концепция реализации

Для реализации используем два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопка)*, которые отображают различные кадры в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* при щелчке (мыши). Настройки прав пользователей выполняются в редакторе *User Administrator (Администратор пользователей)*.

Реализация в проекте WinCC

Шаг	Процедура: Разрешение управления оператором, стандартная панель входа в систему
1	В *проводнике WinCC* откройте редактор *User Administrator (Администратор пользователей)* щелчком R (правой кнопки мыши) на нем и выбором пункта *Open (Открыть)* во всплывающем меню 
2	Используя соответствующую кнопку создайте новую группу пользователей и присвойте ей имя. В данном примере мы используем название *service*.
3	С помощью команды меню *Table (Таблица)* → *Add new Authorization Level (Добавить новый уровень авторизации)* определите уровень авторизации *Picture Change (Изменение кадра)* как в строке 9. Этот уровень авторизации присваивается группе *service*. Для этого выберите группу (мышью). В таблице, содержащей строку *Picture Change (Изменение кадра)*, выберите радио–кнопку в колонке *Authorization (Авторизация)* с помощью D (двойного щелчка мыши). Уровень авторизации, присвоенный группе или пользователю, отмечается красной точкой в колонке *Authorization (Авторизация)*. 

Шаг	Процедура: Разрешение управления оператором, стандартная панель входа в систему
4	С помощью кнопки создайте нового пользователя для группы *service*. В примере проекта создан пользователь по имени *willi* с паролем *Project_CreatePicture*. Активируйте отмечаемый блок *Copy Group Settings Also (Копировать вместе с групповыми установками)* для передачи пользователю групповых уровней авторизации.  С помощью команды меню *File (Файл)* → *Exit (Выход)* закройте редактор *User Administrator (Администратор пользователей)*.
5	В проводнике WinCC щелкните R (правой кнопкой мыши) по имени проекта для открытия доступа к свойствам проекта. В отображаемом окне выберите вкладку *Hotkeys (Горячие клавиши)* и произведите требуемые установки для вызова диалогов входа и выхода из системы. Для назначения горячих клавиш щелкните (мышью) по кнопке *Assign (Назначить)*. В данном примере используются комбинации клавиш CTRL+O для входа и CTRL+F для выхода. 

Шаг	Процедура: Разрешение управления оператором, стандартная панель входа в систему
6	Создайте в кадре два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. В данном примере используются объекты *Button3* и *Button4*. Создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, в который кадры вставляются с помощью *прямых соединений* двух *кнопок*.
7	Для объектов *Button3* и *Button4* выберите уровень авторизации *Picture Change (Изменение кадра)* в поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *User Level (Уровень пользователя)* и установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Operator–Control Enable (Разрешение управления оператором)* в *No (Нет)*.
8	При выделенном объекте *Button3* выберите вкладку *Standard Dynamics (Стандартная динамика)* и затем пункт *Operational if Authorized (Действующий при наличии авторизации)* с помощью D (двойного щелчка мыши) в *динамическом мастере*. Завершите *динамический мастер*, щелкнув по кнопке *Finish (Завершение)*. Повторите эту же процедуру для *Button4*.
9	В *менеджере тегов* создайте системный тег *@CurrentUser* типа *Text Tag 16–Bit Character Set (Текстовый тег 16–битного набора символов)*. Имя текущего пользователя автоматически присваивается этому тегу.
10	Активизируйте *процедуры Си* для *Button3* и *Button4*, созданные *динамическим мастером*, по изменению этого тега. Это означает, что *процедура Си* будет далее выполняться не каждые 2 секунды, а только при изменении имени пользователя.

Процедура Си, созданная динамическим мастером

```
#include "apdefap.h"
BOOL _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN

{
#pragma code ("UseAdmin.DLL")
#include "pwrt_api.h"
#pragma code ()
#define NO_MESSAGEBOX  1
CMN_ERROR err;
DWORD pwlevel = 0;
pwlevel = (DWORD) GetPasswordLevel(lpszPictureName,lpszObjectName);
if (pwlevel==0)
return (TRUE);
else
return(PWRTCheckPermissionOnPicture(pwlevel,lpszPictureName,NO_MESSAGEBOX,&

}
```

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Необходимо скорректировать имена групп, пользователей и пароли.

3.4.3 Разрешение управления оператором, вход в систему с использованием отдельного диалога (example 03)

Постановка задачи

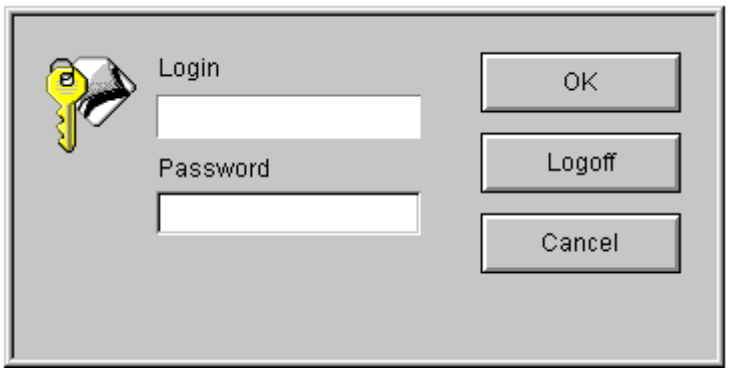
Разрешить выход из режима исполнения по кнопке должен, только если пользователь имеет соответствующую авторизацию. По нажатию кнопки должен отображаться диалог входа в систему.

Концепция реализации

Для реализации будем использовать два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. С помощью первой кнопки при щелчке  (мышью) должно отображаться *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* для входа в систему. Вторая кнопка используется для выхода из режима исполнения.

Реализация в проекте WinCC

Шаг	Процедура: Разрешение управления оператором, вход в систему с использованием отдельного диалога
1	В редакторе *User Administrator (Администратор пользователей)* создайте новую группу пользователей и дайте ей имя. В данном примере используется имя *user*. В строке 10 определите новый уровень авторизации, называющийся *Exit Runtime (Выход из режима исполнения)*. Этот уровень авторизации присваивается созданной группе пользователей. Создайте пользователя для этой группы. В примере проекта создается пользователь с именем *ulrich* и паролем *Project_CreatePicture*.
2	Создайте в кадре два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. В данном примере используются объекты *Button5* и *Button6*.
3	С кнопкой *Button5* свяжите вызов *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* для завершения исполнения. В данном примере используется *окно кадра Picture Window5*.
4	Для объекта *Button5* выберите уровень авторизации *Exit Runtime (Выход из режима исполнения)* в поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *User Level (Уровень пользователя)* и установите атрибут *Property (Свойство)* → *Miscellaneous (Разное)* → *Operator–Control Enable (Разрешение управления оператором) – No (Нет)*.
5	Примените динамический мастер *Operational if Authorized (Действующий при наличии авторизации)* к *Button5*. Установите запуск созданной *процедуры Cи* системным тегом *@CurrentUser*.

Шаг	Процедура: Разрешение управления оператором, вход в систему с использованием отдельного диалога
6	Создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В данном примере используется объект *Picture Window4*. Установите *Property (Свойство)* → *Geometry (Геометрия)* → *Window Width (Ширина окна) – 360* и *Property (Свойство)* → *Geometry (Геометрия)* → *Window Height (Высота окна) – 180*. Установите атрибуты *Properties (Свойство)* → *Miscellaneous (Разное)* → *Moveable*, *Border*, *Title (Перемещаемый, Рамка, Заголовок)* и *Foreground (Передний план)* в *Yes (Да)*. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* выберите кадр *pictu_5_window_18.pdl*. Этот кадр может быть взят прямо из проекта *Project_CreatePicture*. 
7	Для объекта *Button6* создайте *прямое соединение* для отображения созданного *окна кадра*.
8	Для объекта *Button6* создайте *процедуру Си*, которая в зависимости о того, зарегистрирован пользователь или нет, присваивает метке кнопки соответствующий текст. Эта *процедура Си* также вызывается тегом *@CurrentUser*.

Процедура Си для Button6

```
#include "apdefap.h"
 char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropert

{
if (strcmp(GetTagChar("@CurrentUser"),""))
        return "Logoff";
else    return "Logon";
}
```

Если тег *@CurrentUser* содержит имя, т.е. сравнение двух текстов дает результат *TRUE (Истинно)*, возвращается текст *Logoff (Выход из системы)*, в противном случае возвращается текст *Logon (Вход в систему)*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Необходимо скорректировать имена групп, пользователей и пароли.

3.5 Масштабирование кадра

Примеры, относящиеся к этой теме доступны в проекте *Project_CreatePicture* по щелчку (мыши) на изображенной выше кнопке. Примеры приведены в кадре *pictu_3_chapter_04.pdl*.

3.5.1 Переключение геометрии кадра между двумя размерами (example 01)

Постановка задачи

Отображать и скрывать окно кадра с помощью двух кнопок, управляемых мышью. При открытии размер кадра должен быть минимальным. Размер кадра должен регулироваться с помощью двух дополнительных кнопок.

Концепция реализации

Для реализации используем два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*, которые будут отображать и скрывать кадр в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окне кадра)* при нажатии  (мыши). Два дополнительных объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)* увеличивают и уменьшают кадр.

Реализация в проекте WinCC

Шаг	Процедура: Переключение геометрии кадра между двумя размерами
1	Создайте кадр, который следует отображать и скрывать. В примере используется кадр *pictu_3_chapter_00* (начальный кадр проекта *Project_CreatePicture)*.
2	В другом кадре создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, в примере это *Picture Window1*. Установите *Property (Свойство)* → *Geometry (Геометрия)* → *Width (Ширина)* равной *172* и *Property (Свойство)* → *Geometry (Геометрия)* → *Height (Высота) – 140*. Установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Border (Рамка)* в *Yes (Да)* и *Property (Свойство)* → *Miscellaneous (Разное)* → *Adapt Picture (Подгонять размер кадра)* в *Yes (Да)*. Таким образом, кадр, имеющий размеры 859*698, приводится к размерам окна кадра. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* выберите кадр *pictu_3_chapter_00*. Установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Display (Отображение)* в *No (Нет)*.
3	Создайте в том же кадре два дополнительных объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. В данном примере это объекты *Button1* и *Button2*. Для *Button1* создайте *прямое соединение* с *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Соедините *источник Constant (константа)* → *1* с *приемником Object in Picture (Объект кадра)* → *Picture Window1* → *Display (Отображение)*. Внесите изменения, щелкнув по кнопке *OK*.

Шаг	Процедура: Переключение геометрии кадра между двумя размерами
4	Создайте в том же кадре два дополнительных объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. В данном примере это объекты *Button3* и *Button4*. У *Button3* создайте *процедуру Си* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*, которая увеличивает *окно кадра*, скрывает *Button3* и отображает *Button4*. Аналогичным образом у *Button4* для *Events* → *Mouse* → *Press left* создайте *процедуру Си*, которая уменьшает *окно кадра*, скрывает *Button4* и отображает *Button3*. Установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Display (Отображение)* для обеих *кнопок* в *No (Нет)*.
5	У *Button1* создайте *прямое соединение* для *Events (События)* → *Mouse(Мышь)* → *Mouse Action (Действие мыши)*. Соедините *источник Constant (константа)* → *1* с *приемником Object in Picture (Объект кадра)* → *Button3* → *Display (Отображение)*. Подтвердите установки, щелкнув по кнопке *OK*. У *Button2* создайте *процедуру Си* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*, которая скрывает *Button3* и *Button4*, уменьшает размер окна кадра *Picture Window1* и затем скрывает *окно кадра*.
6	Разместите *Button3* и *Button4* друг над другом.

Процедура Си для Button3

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetHeight(lpszPictureName,"Picture Window1",420);
SetWidth(lpszPictureName,"Picture Window1",516);
SetVisible(lpszPictureName,"Button3",0);
SetVisible(lpszPictureName,"Button4",1);
}
```

Измените высоту и ширину *окна кадра Picture Window1* с помощью *внутренних функций SetHeight* и *SetWidth*.

Скройте *кнопку* увеличения (*Button3*).

Отобразите *кнопку* уменьшения (*Button4*).

Процедура Си для Button4

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetHeight(lpszPictureName,"Picture Window1",140);
SetWidth(lpszPictureName,"Picture Window1",172);
SetVisible(lpszPictureName,"Button3",1);
SetVisible(lpszPictureName,"Button4",0);
}
```

Измените высоту и ширину *окна кадра Picture Window1* с помощью *внутренних функций SetHeight* и *SetWidth*.

Отобразите *кнопку* увеличения (*Button3*).

Скройте *кнопку* уменьшения (*Button4*).

Процедура Си для Button2

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetVisible(lpszPictureName,"Button3",0);
SetVisible(lpszPictureName,"Button4",0);
SetHeight(lpszPictureName,"Picture Window1",140);
SetWidth(lpszPictureName,"Picture Window1",172);
SetVisible(lpszPictureName,"Picture Window1",0);
}
```

Спрячьте кнопки увеличения (*Button3*) и уменьшения (*Button4*).

Измените высоту и ширину *окна кадра Picture Window1* с помощью *внутренних функций SetHeight* и *SetWidth*.

Скройте *окно кадра Picture Window1*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Для *прямых соединений* у *Button1* должны быть правильно настроены имена объектов.

В *процедурах Си* для объектов *Button2*, *Button3* и *Button4* необходимо указать имена объектов и устанавливаемые размеры кадров.

3.5.2 Плавное изменение размеров кадра (example 02)

Постановка задачи

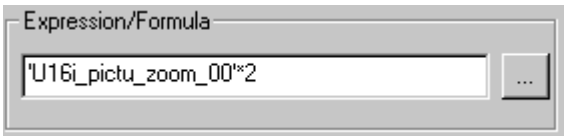
Отображать и скрывать окно кадра при помощи двух кнопок, управляемых мышью. Кроме того, размер кадра должен плавно регулироваться с помощью бегунка.

Концепция реализации

Для реализации будем использовать два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)* для отображения и сокрытия кадра в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окне кадра)* при щелчке (мыши) и *Windows Object (Объект Windows)* → *Slider Object (Бегунок)* для изменения размера кадра.

Реализация в проекте WinCC

Шаг	Процедура: Плавное изменение размеров кадра
1	Создайте кадр, который следует отображать и скрывать. В примере используется кадр *pictu_5_window_10.pdl*, у которого отношение ширины к высоте 2 : 1.
2	Создайте в другом кадре *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В примере это *Picture Window2*. Установите атрибут *Property (Свойство)* → *Geometry (Геометрия)* → *Width (Ширина)* равным *160* и *Property (Свойство)* → *Geometry (Геометрия)* → *Height (Высота) – 80* (отношение высота : ширина по–прежнему 2 : 1). Для отображения в режиме исполнения окна с рамкой, установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Border (Рамка) – Yes (Да)* и *Property (Свойство)* → *Miscellaneous (Разное)* → *Adapt Picture (Подгонять размер кадра) –Yes (Да)*. Таким образом, кадр приводится к размеру *окна кадра*. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* выберите кадр *pictu_5_window_10.pdl*. Установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Display (Отображение) – No (Нет)*.
3	Создайте в том же кадре две дополнительные кнопки. В примере это объекты *Button5* и *Button6*. У *Button5* создайте *прямое соединение* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Соедините *источник Constant (Константа)* → *1* с *приемником Object in Picture (Объект кадра)* → *Picture Window2* → *Display (Отображение)*. Подтвердите изменения, щелкнув по кнопке *OK*.
4	Аналогичным образом создайте *прямое соединение* у *Button6* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. В качестве *Constant (Константа)* укажите значение *0*.
5	В *менеджере тегов* создайте тег типа *Unsigned 16–Bit Value (16–битная величина без знака)*. В данном примере используется тег *U16i_pictu_zoom_00*.

Шаг	Процедура: Плавное изменение размеров кадра
6	Создайте *Windows Object (Объект Windows)* → *Slider Object (Бегунок)*. В примере это *Slider Object1*. Установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Maximum Value (Максимальная величина) – 300*. Установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Process Driver Connection (Соединение с драйвером процесса) – 80*. В поле *Events (События)* → *Property Topics (Разделы свойств)* → *Miscellaneous (Разное)* → *Process Driver Connection (Соединение с драйвером процесса)*, создайте *прямое соединение*. Соедините *источник Property (Свойство)* → *this object (этот объект)* → *Process Driver Connection (Соединение с драйвером процесса)* с *приемником Variable (Переменная)* → *U16i_pictu_zoom_00*. Подтвердите изменения, щелкнув по кнопке *OK*.
7	У объекта *Picture Window2* создайте *динамический диалог* для *Properties (Свойства)* → *Geometry (Геометрия)* → *Window Height (Высота окна)*. Используйте *кнопку*  для выбора *tag (тег)* → *U16i_pictu_zoom_00*. Используйте *кнопку*  в диалоге *Change Trigger (Изменение триггера)* для подтверждения тега *U16i_pictu_zoom_00* в качестве имени триггера и установите стандартный цикл в состояние *Upon Change (По изменению)*. Подтвердите установки, щелкнув по кнопке *OK*. В поле *Data Type (Тип данных)* выберите вариант *Direct (Прямой)* и выйдите из *динамического диалога*, щелкнув по кнопке *Apply (Применить)*.
8	У объекта *Picture Window2* создайте *динамический диалог* для *Properties (Свойства)* → *Geometry (Геометрия)* → *Window Height (Высота окна)*. Установки могут быть сделаны описанным выше способом, однако поле *Expression/Formula (Выражение/Формула)* должно быть заполнено следующим образом:  Это присваивает высоте окна величину, равную удвоенной ширине окна.
9	У объекта *pictu_3_chapter_04* для *Events (События)* → *Miscellaneous (Разное)* → *Open Picture (Открытие кадра)* создайте *процедуру Си*, которая устанавливает величину тега *U16i_pictu_zoom_00* равной 80, при открытии кадра. Без этой инициализации величина тега останется равной 0 до первого использования *Slider Object1*. Если затем будет нажата кнопка *Button5*, кадр *Picture Window2* будет отображен с размерами 0x0.

Процедура Си для открытия кадра

```
#include "apdefap.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpszPr

{
//init tag
SetTagWord("U16i_pictu_zoom_00",80);
}
```

Установите величину тега *U16i_pictu_zoom_00* равной *80*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Для *прямого соединения* в *Slider Object1* следует задать соответствующее имя тега.

Для *динамических диалогов* в объекте *Picture Window2* следует указать соответствующие имена тегов. Для используемого отношения ширина : высота должен быть задан соответствующий множитель.

3.5.3 Создание настраиваемого кадра с использованием диалога свойств (example 03)

Постановка задачи

Предоставить возможность растягивания *окна кадра* мышью до любого размера. Кроме этого кадр должен перемещаться в любое положение на экране. Необходимо обеспечить возможность увеличения размеров кадра до максимума и его сокрытия с помощью *кнопки*.

Концепция реализации

Для реализации используем два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*, которые будут отображать и прятать кадр в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, по щелчку  (мыши). Необходимые свойства кадра задаются в диалоговом окне свойств.

Реализация в проекте WinCC

Шаг	Процедура: Создание настраиваемого кадра с использованием диалога свойств
1	Создайте кадр, который следует отображать и скрывать. В примере используется кадр pictu_3_chapter_00 picture (начальный кадр проекта *Project_CreatePicture)*.
2	Создайте в другом кадре *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, в примере это *Picture Window3*. Установите *Property (Свойство)* → *Geometry (Геометрия)* → *Width (Ширина)* – *147* и *Property (Свойство)* → *Geometry (Геометрия)* → *Height (Высота) – 140*. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* установите атрибуты *Sizeable (Изменяемый размер)*, *Moveable (Перемещаемый)*, *Border (Рамка)*, *Title (Заголовок)*, *Can Be Maximized (Может быть увеличен до максимума)*, *Adapt Picture (Подгонка размера)* и *Can Be Closed (Может быть закрыт)* в *Yes (Да)*. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* выберите кадр *pictu_3_chapter_00*. Установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Display (Отображение)* в *No (Нет)*.
3	Создайте в том же кадре два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. В данном примере используются объекты *Button7* и *Button8*. У *Button7* создайте *прямое соединение* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Соедините *источник Constant (Константа)* → *1* с *приемником Object in Picture (Объект кадра)* → *Picture Window3* → *Display (Отображение)*. Подтвердите установки, щелкнув по кнопке *OK*.
4	Аналогичным образом создайте *прямое соединение* у *Button8* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Однако в качестве константы введите *0*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Для *прямых соединений* в объектах *Button7* и *Button8* должны быть установлены соответствующие имена отображаемого кадра *окна кадра*.

Кадр, отображаемый в *окне кадра Picture Window3*, следует модифицировать.

3.6 Элементы управления

Примеры, относящиеся к данной теме доступны в проекте *Project_CreatePicture* по щелчку (мыши) на изображенной выше кнопке. Примеры приведены в кадре *pictu_3_chapter_05.pdl*.

3.6.1 Двоичная операция переключения (двухступенчатое управление) (example 01)

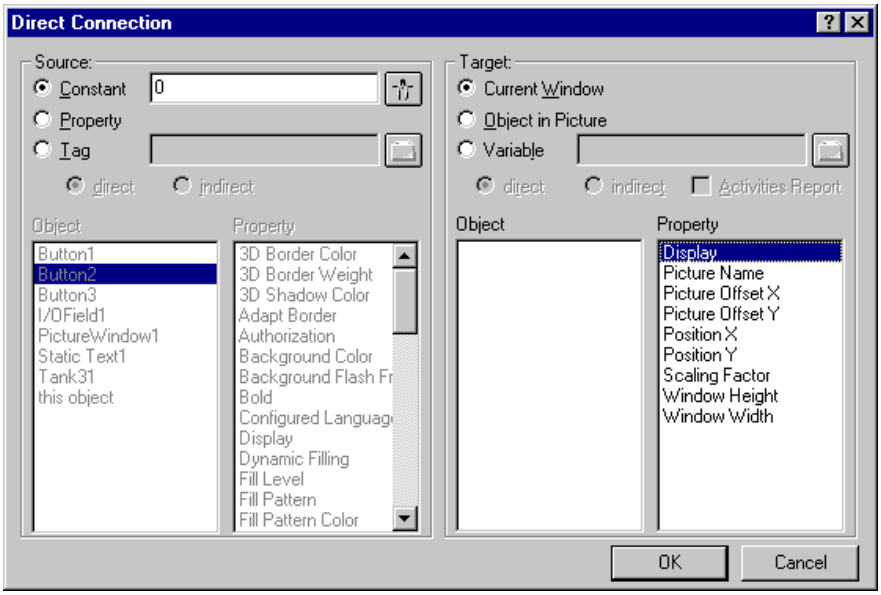
Постановка задачи

Организовать доступ к панели управления с помощью *кнопки*, управляемой мышью. Одна *кнопка* панели должна включать и выключать клапан, а другая — закрывать панель.

Концепция реализации

Для реализации будем использовать объект *Windows Object (Объект Windows)* → *Button (Кнопка)*, который будет отображать кадр в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, при щелчке (мыши) и две дополнительные кнопки, выполняющие операцию переключения и закрытие панели.

Реализация в проекте WinCC

Шаг	Процедура: Двоичная операция переключения (двухступенчатое управление)
1	В *менеджере тегов* создайте тег типа *Binary Tag (Двоичный тег)*. В примере используется тег *BINi_pictu_input_00*. Этот тег содержит текущее состояние клапана.
2	Создайте кадр с двумя объектами *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. В примере используется кадр *pictu_5_window_11*, содержащий объекты *Button1* и *Button2*. Создайте у *Button1 прямое соединение* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Соедините *источник Constant (Константа)* → *0* с приемником *Current Window (Текущее окно)* → *Display (Отображение)*. Подтвердите установки щелчком по кнопке *OK*. 
3	У второй кнопки *Button2* создайте *процедуру Cu*, которая инвертирует двоичный тег *BINi_pictu_input_00*.

Шаг	Процедура: Двоичная операция переключения (двухступенчатое управление)
4	В другом кадре создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, в примере это *Picture Window1*. Установите *Property (Свойство)* → *Geometry (Геометрия)* → *Width (Ширина) – 246* и *Property (Свойство)* → *Geometry (Геометрия)* → *Height (Высота) – 129*. Для отображения окна с рамкой и перемещения его во время выполнения установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Border (Рамка)* в *Yes (Да)* и *Property (Свойство)* → *Miscellaneous (Разное)* → *Moveable (Перемещаемый)* в *Yes (Да)*. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* выберите кадр *pictu_3_window_11.pdl*. Установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Display (Отображение)* в *No (Нет)*.
5	Создайте в кадре объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере это объект *Button1* в кадре *pictu_3_chapter_05.pdl*. У *Button1* создайте *прямое соединение* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Соедините *источник Constant (Константа)* → *1* с приемником *Object in Picture(Объект кадра)* → *Picture Window1* → *Display (Отображение)*. Подтвердите установки щелчком по кнопке *OK*.

Процедура Си для Button2

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetTagBit("BINi_pictu_input_00", (SHORT)!GetTagBit("BINi_pictu_input_00"));
}
```

Состояние тега *BINi_pictu_input_00* считывается и инвертируется при помощи *внутренней функции GetTagBit*, после чего повторно устанавливается *внутренней функцией SetTagBit*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Для *прямого соединения* в *Button1* следует скорректировать имя открываемого *окна кадра*.

В *процедуре Си* у *Button2* панели оператора необходимо изменить имя тега.

3.6.2 Операция двоичного переключения S–R (двухступенчатое управление) (example 02)

Постановка задачи

Организовать доступ к панели управления с помощью *кнопки*, управляемой мышью. Одна *кнопка* панели должна включать клапан, а другая — выключать его. Панель должна закрываться щелчком по отдельной *кнопке*.

Концепция реализации

Для реализации будем использовать объект *Windows Object (Объект Windows)* → *Button (Кнопка)*, который будет отображать кадр в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* при щелчке  (мыши) и три дополнительные *кнопки*, выполняющие операции переключения и закрытия панели.

Реализация в проекте WinCC

Шаг	Процедура: Операция двоичного переключения S–R (двухступенчатое управление)
1	В *менеджере тегов* создайте тег типа *Binary Tag (Двоичный тег)*. В примере используется тег *BINi_pictu_input_01*. Этот тег содержит текущее состояние клапана.
2	Создайте кадр с тремя объектами *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. В примере используется кадр *pictu_5_window_12*, включающий объекты *Button1*, *Button2* и *Button3*. Создайте у *Button1 прямое соединение* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Соедините *источник Constant (Константа)* → *0* с приемником *Current Window (Текущее окно)* → *Display (Отображение)*. Подтвердите установки щелчком по кнопке *OK*.
3	Создайте у *Button2 прямое соединение* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Соедините *источник Constant (Константа)* → *1* с приемником *Variable (Переменная)* → *BINi_pictu_input_01*. Подтвердите установки щелчком по кнопке *OK*.
4	Аналогичным образом создайте *прямое соединение* для *Button3* в *Events* → *Mouse* → *Press Left*. В качестве *Constant*, укажите величину *0*.
5	В другом кадре создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, в примере это *Picture Window2*. Установите *Property (Свойство)* → *Geometry (Геометрия)* → *Width (Ширина) – 246* и *Property (Свойство)* → *Geometry (Геометрия)* → *Height (Высота) – 129*. Для отображения окна с рамкой и перемещения его во время выполнения установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Border (Рамка)* в *Yes (Да)* и *Property (Свойство)* → *Miscellaneous (Разное)* → *Moveable (Перемещаемый)* в *Yes (Да)*. В *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* выберите кадр *pictu_3_window_12.pdl*.

Шаг	Процедура: Операция двоичного переключения S–R (двухступенчатое управление)
6	Создайте в том же кадре объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере это объект *Button2* в кадре *pictu_3_chapter_05.pdl*. У *Button2* создайте *прямое соединение* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Соедините *источник Constant (Константа)* → *1* с приемником *Object in Picture(Объект кадра)* → *Picture Window1* → *Display (Отображение)*. Подтвердите установки щелчком по кнопке *OK*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

У *прямого соединения* для *Button2* следует изменить имя открываемого *окна кадра*.

У *прямых соединений* для *Button1* и *Button2* панели оператора необходимо модифицировать имена тегов.

3.6.3 Операция двоичного переключения с подтверждением (example 03)

Постановка задачи

Организовать доступ к панели управления с помощью *кнопки*, управляемой мышью. Одна *кнопка* панели должна включать клапан, а другая — выключать его. Фактическая операция переключения должна происходить только после нажатия отдельной кнопки *OK*, закрывающей управляющую панель.

Концепция реализации

Для реализации будем использовать объект *Windows Object (Объект Windows)* → *Button (Кнопка)*, который будет отображать кадр в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* при щелчке  (мыши) и две дополнительные *кнопки*, выполняющие операцию переключения и закрытие панели.

Реализация в проекте WinCC

Шаг	Процедура: Операция двоичного переключения с подтверждением
1	В *менеджере тегов* создайте два тега типа *Binary Tag (Двоичный тег)*. В примере используются теги *BINi_pictu_input_02* и *BINi_pictu_input_03*. *BINi_pictu_input_02* содержит текущее состояние клапана, *BINi_pictu_input_03* служит в качестве буфера для операции переключения перед подтверждением.
2	Создайте кадр с двумя объектами *Windows Object (Объект Windows)* → *Button (Кнопка)*. В примере используется кадр *pictu_5_window_13.pdl*, включающий объекты *Button1* и *Button2*. У *Button1* создайте *прямое соединение* для *Events (События)* → *Mouse Action (Действие мыши)*. Соедините *источник Constant (Константа)* → *0* с *приемником Current Window (Текущее окно)* → *Display (Отображение)*. Подтвердите установки щелчком по кнопке *OK*. Создайте другое *прямое соединение* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Соедините *источник BINi_pictu_input_02* с *приемником BINi_pictu_input_03*. Подтвердите установки щелчком по кнопке *OK*.
3	Для второй кнопки *Button2* создайте *процедуру Cu*, инвертирующую двоичный тег *BINi_pictu_input_02*.
4	В другом кадре — *pictu_3_chapter_05.pdl* — создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В данном примере это объект *Picture Window3*. Установите *Property (Свойство)* → *Geometry (Геометрия)* → *Width (Ширина) – 246* и *Property (Свойство)* → *Geometry (Геометрия)* → *Height (Высота) – 129*. Для отображения окна с рамкой и перемещения его во время выполнения установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Border (Рамка)* в *Yes (Да)* и *Property (Свойство)* → *Miscellaneous (Разное)* → *Moveable (Перемещаемый)* в *Yes (Да)*. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* выберите кадр *pictu_3_window_13.pdl*.

Шаг	Процедура: Операция двоичного переключения с подтверждением
5	Создайте в том же кадре объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере это объект *Button3* в кадре *pictu_3_chapter_05.pdl*. У *Button3* создайте *прямое соединение* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Соедините *источник Constant (Константа)* → *1* с приемником *Object in Picture (Объект кадра)* → *Picture Window3* → *Display (Отображение)*. Подтвердите установки щелчком по кнопке *OK*

Процедура Си для Button2

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetTagBit("BINi_pictu_input_02",(SHORT)!GetTagBit("BINi_pictu_input_02"));
}
```

Состояние тега *BINi_pictu_input_02* считывается и инвертируется при помощи *внутренней функции GetTagBit*, после чего повторно устанавливается *внутренней функцией SetTagBit*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

У *прямого соединения* в *Button3* следует изменить имя открываемого *окна кадра*.

У *прямых соединений* в *Button1* панели оператора необходимо модифицировать имена тегов.

В *процедуре Си* для *Button2* следует изменить имя тега.

3.6.4 Автоматическая проверка ввода (example 04)

Примеры, относящиеся к данной теме, доступны в проекте *Project_CreatePicture* по щелчку (мыши) на изображенной выше кнопке. Пример приведен в кадре *pictu_3_chapter_05a.pdl*.

Постановка задачи

Организовать доступ к панели управления с помощью *кнопки*, управляемой мышью. Панель должна использоваться для задания количества жидкости, заполняющей контейнер. Вводимая величина должна автоматически проверяться на предмет превышения максимального уровня заполнения контейнера.

Концепция реализации

Для реализации будем использовать объект *Windows Object (Объект Windows)* → *Button (Кнопка)*, который будет отображать кадр в *Smart Object (Интеллектуальный объект)* → *Picture Window(Окно кадра)*, при щелчке (мыши). Кроме того, будем использовать три объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)* для включения и выключения клапана и закрытия панели оператора. Объект *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)* будем использовать для ввода уровня заполнения.

Реализация в проекте WinCC

Шаг	Процедура: Автоматическая проверка ввода
1	В *менеджере тегов* создайте тег типа *Binary Tag (Двоичный тег)*, содержащий текущее состояние клапана. В примере используется тег *BINi_pictu_input_06*.
2	Создайте два тега типа *Unsigned 16–Bit Value (16–битная величина без знака)*. В данном примере это теги *U16i_pictu_input_04* и *U16i_pictu_input_05*. Первый из них содержит уставку уровня заполнения контейнера, второй — фактическую величину.
3	Создайте кадр с тремя объектами *Windows Objects (Объект Windows)* → *Buttons (Кнопки)* и *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В примере используются объекты *Button1*, *Button2*, *Button3* и *I/O Field1*. В качестве кадра используется *pictu_5_window_14.pdl*.
4	В *диалоге конфигурирования* объекта *I/O Field1* создайте *соединение с тегом U16i_pictu_input_04* и установите срабатывание по изменению.
5	Будем считать, что контейнер имеет максимальный уровень заполнения 40 литров. Следовательно, *поле ввода/вывода* должно допускать ввод значений между 0 и 40. Для этого установите *Property (Свойство)* → *Limits (Пределы)* → *Low Limit Value (Нижний предел величины) – 0* и *Property (Свойство)* → *Limits (Пределы)* → *High Limit Value (Верхний предел величины) – 40*.
6	У *Button1* создайте *прямое соединение* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*, которое скрывает кадр.
7	Для *Button1*, создайте *прямое соединение* в *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*, которое присваивает величину *1* тегу *BINi_pictu_input_06*. Для *Button3*, создайте *прямое соединение,* которое присваивает тегу величину 0.

Шаг	Процедура: Автоматическая проверка ввода
8	Создайте во втором кадре *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В данном примере используется объект *Picture Window1*. Задайте размеры *окна кадра* так, чтобы они совпадали с размерами созданного кадра. Если *окно кадра* следует отображать с рамкой, *Height (Высота)* и *Width (Ширина) окна кадра* должны быть на 10 пикселей больше, чем те же величины кадра. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* выберите кадр *pictu_5_window_14.pdl*.
9	Создайте в том же кадре объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В примере это объект *Button1*. Для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте прямое соединение. Соедините *источник Constant (Константа)* → *1* с приемником *Object in Picture (Объект кадра)* → *Picture Window1* → *Display (Отображение)*. Примените установки, щелкнув по кнопке *OK*.
10	Для отображения уровня заполнения использован библиотечный объект *Tank2*. С целью имитации процесса заполнения создана *процедура Си* для *Properties (Свойства)* → *Geometry (Геометрия)* → *Width (Ширина)*. В поле *Properties (Свойства)* → *Tag Assignment (Назначение тега)* → *Fill Level (Уровень заполнения)* создано *соединение с тегом U16i_pictu_input_05*.
11	Для второй формы отображения уровня заполнения использован *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)* — в примере это *I/O Field1*.

Процедура Си для имитации процесса заполнения

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty

{
BOOL state;
SHORT level1,level2;

//get valve state
state=GetTagBit("BINi_pictu_input_06");

if (state==TRUE) {
        level1=GetTagWord("U16i_pictu_input_04");
        level2=GetTagWord("U16i_pictu_input_05");
        level2++;
        if (level2>=level1) {
                SetTagBit("BINi_pictu_input_06",FALSE);
                }//if
        if (level2<=level1) {
                SetTagWord("U16i_pictu_input_05",level2);
                }//if
        }//if
return(80);
}
```

Чтение состояния клапана.

Когда клапан открыт, считываются уставка и фактическая величина уровня заполнения. Инкрементируется фактическую величину. Когда она достигнет уровня уставки, клапан закрывается. Обновляется значение тега, содержащего фактическую величину.

Возвращаемая величина — ширина объекта.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

В кадре *pictu_5_window_14.pdl* имена тегов и пределы *поля ввода/вывода* должны быть установлены в соответствие с вашими требованиями.

3.6.5 Расширенная автоматическая проверка ввода (example 05)

Примеры, относящиеся к данной теме, доступны в проекте *Project_CreatePicture* по щелчку (мыши) на изображенной выше кнопке. Пример приведен в кадре *pictu_3_chapter_05a.pdl*.

Постановка задачи

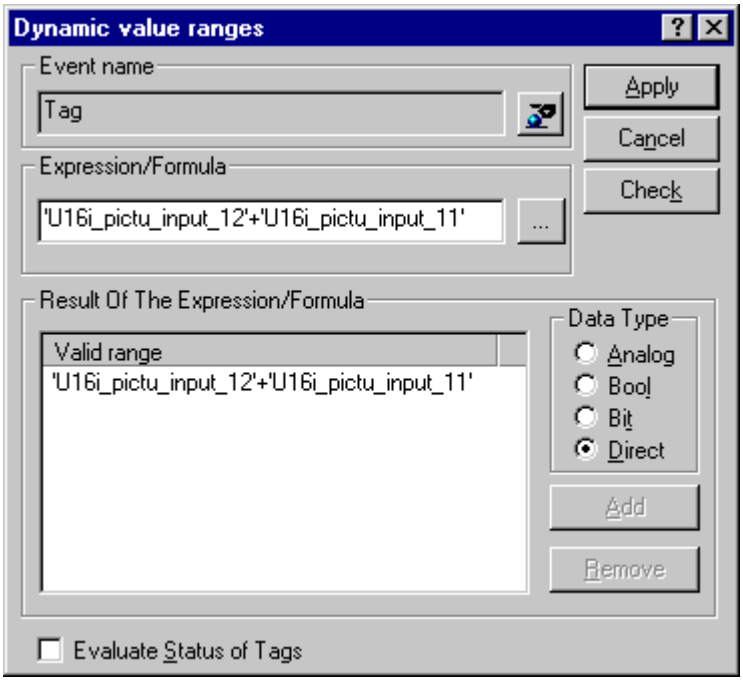
Организовать доступ к панели управления с помощью *кнопки*, управляемой мышью. Панель должна использоваться для, заполнения контейнера двумя жидкостями в определенной пропорции. Сумма двух вводимых величин должна автоматически проверяться на предмет превышения максимального уровня заполнения контейнера.

Концепция реализации

Для реализации будем использовать объект *Windows Object (Объект Windows)* → *Button (Кнопка)*, который будет отображать кадр в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, при щелчке (мыши). Три объекта *Smart Objects (Интеллектуальный объект)* → *I/O Fields (Поля ввода/вывода)* используются для ввода величин заполнения. Кроме того, будем использовать два объекта *Windows Object (Объект Windows)* → *Button (Кнопка)* для принятия сделанных в *полях ввода/вывода* установок или отказа от них.

Реализация в проекте WinCC

Шаг	Процедура: Расширенная автоматическая проверка ввода
1	В *менеджере тегов* создайте два тега типа *Binary Tag (Двоичный тег)*, содержащих текущие состояния клапанов, используемых для заполнения контейнера. В примере используются теги *BINi_pictu_input_09* и *BINi_pictu_input_10*.
2	Создайте четыре тега типа *Unsigned 16–Bit Value (16–битная величина без знака)*. В данном примере это теги *U16i_pictu_input_07*, *U16i_pictu_input_08*, *U16i_pictu_input_13* и *U16i_pictu_input_14*. Первые два содержат уставки уровней заполнения контейнера, а последние два — фактические величины.
3	Создайте два тега типа *Unsigned 16–Bit Value (16–битная величина без знака)*. В примере это теги *U16i_pictu_input_11* и *U16i_pictu_input_12*. Они содержат величины, введенные в *полях ввода/вывода*.
4	Создайте кадр с двумя объектами *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)* и тремя *Smart Objects (Интеллектуальные объекты)* → *I/O Fields (Поля ввода/вывода)*. В данном примере используются объекты *Button1*, *Button2* и *I/O Field1*, *I/O Field2*, *I/O Field3*. В качестве кадра используется *pictu_5_window_15.pdl*
5	В *диалоге конфигурирования* объекта *I/O Field1* создайте *соединение с тегом U16i_pictu_input_11* и установите срабатывание по изменению. Для *I/O Field2* создайте *соединение с тегом U16i_pictu_input_12*.

Шаг	Процедура: Расширенная автоматическая проверка ввода
6	У *I/O Field3* создайте *динамический диалог* для *Properties (Свойства)* → *Output/Input (Вывод/ввод)* → *Output Value (Выводимая величина)*. Введите установки, показанные на следующем рисунке. Установите срабатывание по изменению. 
7	У *Button2* создайте *прямое соединение* для *Events (События)* → *Mouse (мышь)* → *Press Left (Нажатие левой кнопки)*, которое скрывает кадр.
8	У *Button1* создайте *процедуру Си* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*, которая присваивает содержимое тегов ввода *U16i_pictu_input_11* и *U16i_pictu_input_12* тегам уставок *U16i_pictu_input_07* и *U16i_pictu_input_08*. Для *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* создайте *прямое соединение*, которое скрывает кадр.
9	Создайте в том же кадре два *Standard Objects (Стандартные объекты)* → *Static Texts (Статические тексты)*. В примере используются объекты *Static Text5* и *Static Text6*. Они используются для индикации превышения максимального уровня. В объекте *Static Text5*, который содержит сообщение об ошибке, установите *Property (Свойство)* → *Miscellaneous(Разное)* → *Display (Отображение)* в *No(нет)*.
10	У *I/O Field3* создайте *процедуру Си* для *Events (События)* → *Property Topics (Разделы свойств)* → *Output/Input (Вывод/Ввод)* → *Output Value (Выводимая величина)*, которая активизирует *Button1,* только если максимальный уровень заполнения не был превышен и отображает текст сообщения об ошибке в противном случае.

Шаг	Процедура: Расширенная автоматическая проверка ввода
11	Создайте во втором кадре *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В примере используется *Picture Window2*. Установите размеры *окна кадра* так, чтобы они соответствовали размерам созданного кадра. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* введите *pictu_5_window_15.pdl*.
12	Создайте в том же кадре объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется объект *Button3*. Для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте прямое соединение. Соедините *источник Constant (Константа)* → *1* с приемником *Object in Picture (Объект кадра)* → *Picture Window2* → *Display (Отображение)*. Подтвердите установки щелчком по кнопке *OK*.
13	Для отображения уровня заполнения используется библиотечный объект *Tank2*. Для имитации процесса заполнения созданы *процедуры Си* в *Properties (Свойства)* → *Geometry (Геометрия)* → *Width (Ширина)* и в *Properties (Свойства)* → *Geometry (Геометрия)* → *Height (Высота)*. Для атрибута *Properties (Свойства)* → *Tag Assignment (Назначение тега)* → *Fill Level (Уровень заполнения)* создан *динамический диалог*, возвращающий сумму двух тегов фактических величин *U16i_pictu_input_13* и *U16i_pictu_input_14*.
14	Для второй формы отображения уровня заполнения используется *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)* — в примере это *I/O Field2*.

Процедура Си для Button1

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF

{
SHORT tmp1,tmp2;

tmp1=GetTagWord("U16i_pictu_input_11");
tmp2=GetTagWord("U16i_pictu_input_12");

if (tmp1>GetTagWord("U16i_pictu_input_07")){
        SetTagWord("U16i_pictu_input_07",tmp1);
        SetTagBit("BINi_pictu_input_09",TRUE);
        }//if
if (tmp2>GetTagWord("U16i_pictu_input_08")){
        SetTagWord("U16i_pictu_input_08",tmp2);
        SetTagBit("BINi_pictu_input_10",TRUE);
        }//if
}
```

Чтение величин тегов, которые были ведены в *полях ввода/вывода*.

Если веденное значение превышает величину текущей уставки, оно запоминается в качестве уставки и включается клапан.

C–Action at I/O Field3

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char*
{
int a;

a=GetTagWord("U16i_pictu_input_11")+GetTagWord("U16i_pictu_input_12");
if (a<=40) {
      SetOperation(lpszPictureName,"Button1",1);
      SetVisible(lpszPictureName,"Static Text5",0);
      SetVisible(lpszPictureName,"Static Text6",1);
      }//if
else {
      SetOperation(lpszPictureName,"Button1",0);
      SetVisible(lpszPictureName,"Static Text5",1);
      SetVisible(lpszPictureName,"Static Text6",0);
      }//if
}
```

Чтение величин тегов, которые были ведены в *полях ввода/вывода*.

Если сумма введенных величин превышает максимальный уровень заполнения контейнера, *Button1* становится неактивной и отображается объект *Static Text5*, содержащий сообщение об ошибке.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

В кадре *pictu_5_window_15.pdl* имена тегов и пределы *поля ввода/вывода* должны быть установлены в соответствии с вашими требованиями.

3.6.6 Множественное применение (example 06)

Примеры, относящиеся к данной теме, доступны в проекте *Project_CreatePicture* по щелчку (мыши) на изображенной выше кнопке. Пример приведен в кадре *pictu_3_chapter_05b.pdl*.

Постановка задачи

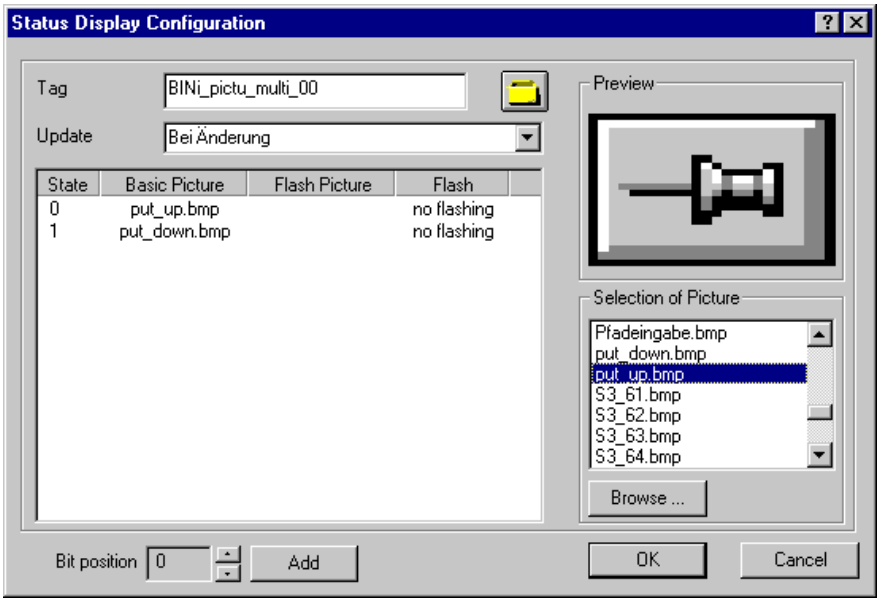
Организовать доступ к панели управления с помощью нескольких *кнопок*, управляемых мышью. *Окно кадра*, открытое определенной *кнопкой*, должно управлять соответствующим ей клапаном. По умолчанию окно оператора открывается рядом с кнопкой, использованной для вызова. Однако оно может быть привязано к любой другой точке.

Концепция реализации

Для реализации будем использовать объекты *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*, которые будут отображать кадр в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)* при щелчке (мыши). Два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)* используются для управления клапаном, дополнительная кнопка — для закрытия окна. Имя клапана и его состояние отображаются с помощью двух объектов *Standard Objects (Стандартные объекты)* → *Static Texts (Статические тексты)*. Привязка кадра осуществляется посредством *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*.

Реализация в проекте WinCC

Шаг	Процедура: Множественное применение
1	В *менеджере тегов* создайте теги типа *Binary Tag (Двоичный тег)*, которые отображают текущие состояния клапанов. Требуемое число тегов зависит от числа клапанов. В примере используются теги *BINi_pictu_multi_01*, *BINi_pictu_multi_02*, *BINi_pictu_multi_03* и *BINi_pictu_multi_04*.
2	Создайте тег типа *Text Tag 16–Bit Character Set (Текстовый тег 16–битного набора символов)*. В примере это тег *T16x_pictu_input_15*. Он может быть использован в роли адресного тега.
3	Создайте тег типа *Binary Tag (Двоичный тег)*. В примере это тег *BINi_pictu_multi_00*. Он содержит информацию о том, привязано ли окно.
4	Создайте кадр с тремя объектами *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. В примере используются объекты *Button1*, *Button2* и *Button3*. В качестве кадра используется *pictu_5_window_16.pdl*.
5	У *Button1* создайте *процедуру Си* для *Events (События)* → *Mouse (Мышь)* → *Press left (Нажатие левой кнопки)*, которая устанавливает положение кадра вне видимого участка, закрывает кадр и отключает привязку кадра.

Шаг	Процедура: Множественное применение
6	У *Button2* создайте *прямое соединение* для *Events (События)* → *Mouse (Мышь)* → *Press left (Нажатие левой кнопки)*. Соедините *источник Constant (Константа)* → *1* с *приемником Variable (Переменная)* → *T16x_pictu_input_15*. Выберите опцию *indirect (косвенная)*. Подтвердите установки щелчком по кнопке *OK*. Так устанавливается косвенная адресация. Аналогичным образом создайте прямое соединение для *Button2* с *источником Constant* → *0*.
7	Создайте *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*. В примере используется *Status Display1*. В следующем *диалоге конфигурирования* выберите тег *BINi_pictu_multi_00* и установите срабатывание по изменению. Используйте *кнопку Add (Добавить)* для того чтобы добавить состояние. Для состояния *0* выберите рисунок *put_up.gif*, а для состояния *1* — рисунок *put_down.gif*. 
8	У *Status Display1* создайте *процедуру Cu* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*, которая инвертирует тег *BINi_pictu_multi_00*.
9	Для заголовка создайте *Standard Object (Стандартный объект)* → *Static Text (Статический текст)*. В данном примере используется объект *Static Text1*. Для *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* создайте *процедуру Cu*, считывающую текущий номер клапана из адресного тега *T16x_pictu_input_15* и возвращающую соответствующий текст.
10	Создайте другой *Standard Object (Стандартный объект)* → *Static Text (Статический текст)* для отображения состояния клапана. В примере используется объект *Static Text2*. Для *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* создайте *процедуру Cu*, считывающую состояние текущего клапана и возвращающую соответствующий текст. Для *Properties (Свойства)* → *Colors (Цвета)* → *Font Color (Цвет шрифта)* создайте *процедуру Cu*, управляющую цветом шрифта в соответствии с состоянием текущего клапана.

Шаг	Процедура: Множественное применение
11	Во втором кадре создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В данном примере используется объект *Picture Window1*. Установите размеры *окна кадра* так, чтобы он они соответствовали размерам кадра. Установите *Properties (Свойства)* → *Miscellaneous (Разное)* → *Moveable (Перемещаемый)* и *Border (Рамка)* в *Yes(Да)*. В *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* укажите кадр *pictu_5_window_16.pdl*.
12	В том же кадре создайте объект *Windows Object (Объект Windows)* → *Button (Кнопка)* для каждого клапана, в примере используются объекты *Button1*, *Button2*, *Button3* и *Button4*. Для каждой *кнопки* создайте *процедуру Cи*, считывающую номер кнопки и записывающую в адресный тег соответствующее имя. В зависимости от того, привязан кадр или нет, он может располагаться справа от вызвавшей кнопки.

Процедура Си для кнопки закрытия (Button1)

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetLeft("pictu_3_chapter_05b","Picture Window1",-1000);
SetVisible("pictu_3_chapter_05b","Picture Window1",0);
SetTagBit("BINi_pictu_multi_00",FALSE);
}
```

Установите положение кадра вне видимого участка.

Спрячьте кадр.

Отключите привязку рисунка.

Процедура Си для Status Display1

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
SetTagBit("BINi_pictu_multi_00",(SHORT)!GetTagBit("BINi_pictu_multi_00"));
}
```

Инвертируйте состояние тега для привязки кадра.

Процедура Си для кнопок управления клапаном

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP

{
int x,y;
char name[20];
int number;
int  ch = n;
char *pdest;
//check if odject name contains caracter n
pdest = strrchr( lpszObjectName, ch );
if ( pdest == NULL )(printf("ObjectNameError"));
//read object number
else {
   number = atoi(strrchr(lpszObjectName,n)+1);
   sprintf(name,"BINi_pictu_multi_%02d",number);
   //generate tag name
   SetTagChar("T16x_pictu_input_15",name);
   }
SetVisible(lpszPictureName,"Picture Window1",1);
if (GetTagBit("BINi_pictu_multi_00")==FALSE){
      //get object position
      y=GetTop(lpszPictureName,lpszObjectName);
      x=GetLeft(lpszPictureName,lpszObjectName);
      //set position of picture window
      SetLeft(lpszPictureName,"Picture Window1",-1000);
      SetTop(lpszPictureName,"Picture Window1",y);
      SetLeft(lpszPictureName,"Picture Window1",(x+22));
      }
}
```

Прочитайте номер объекта в имени объекта.

Сгенерируйте имя тега текущего состояния.

Установите адресный тег в соответствии с тегом текущего состояния.

Отобразите *окно кадра*.

Если *окно кадра* не было привязано, определите положение *кнопки* и установите положение кадра справа от *кнопки*. *Окно кадра* устанавливается вне видимого участка для того чтобы избежать непродолжительного отображения кадра при первом изменении положения.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Установите имена тега и объекта в соответствии с вашими требованиями. Убедитесь в соблюдении соглашений в отношении имен. Номер кнопки должен быть должен быть однозначно связан с именем переключаемого тега.

3.7 Добавление динамики

Примеры, относящиеся к данной теме доступны в проекте *Project_CreatePicture* по щелчку (мыши) на изображенной выше кнопке. Примеры приведены в кадре *pictu_3_chapter_06.pdl*.

3.7.1 Изменение цвета (example 01)

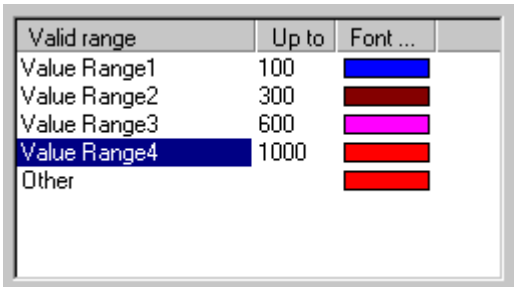
Постановка задачи

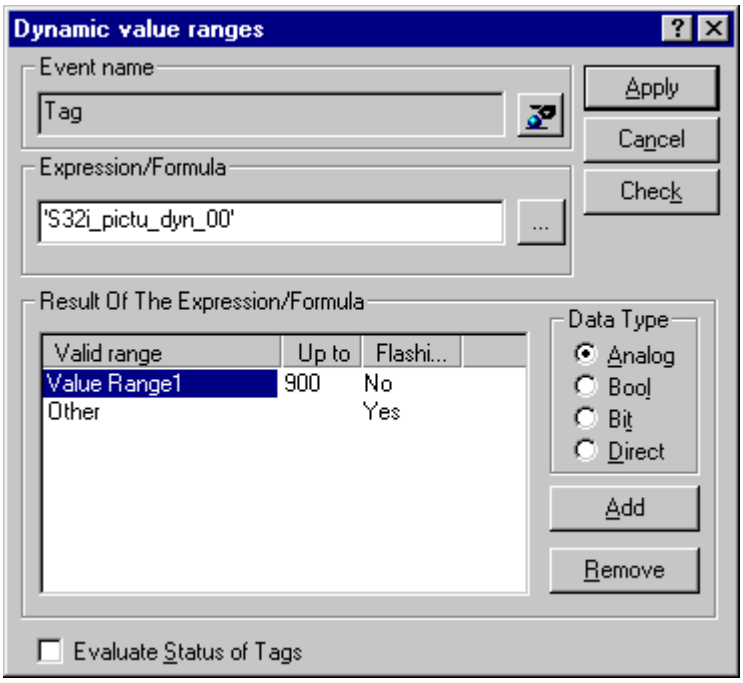
Изменять цвет текста в зависимости от значения тега.

Концепция реализации

Для реализации будем использовать *Windows Object (Объект Windows)* → *Slider Object (Бегунок)*, изменяющий значение тега. Отображение текста реализуется с использованием *Standard Object (Стандартный объект)* → *Static Text (Статический текст)*.

Реализация в проекте WinCC

Шаг	Процедура: Изменение цвета
1	В *менеджере тегов* создайте тег типа *Signed 32–Bit Value (32–битная величина со знаком)*. В примере это тег *S32i_pictu_dyn_00*.
2	Создайте *Windows Object (Объект Windows)* → *Slider Object (Бегунок)*. В данном примере используется объект *Slider Object1*. В диалоге конфигурирования установите *Maximum Value (Максимальная величина)* – *1000* и *Minimum Value (Минимальная величина)* – *0*. Для *Events (События)* → *Property Topics (Разделы свойств)* → *Miscellaneous (Разное)* → *Process Driver Connection (Соединение с драйвером процесса)* создайте *прямое соединение* с тегом *S32i_pictu_dyn_00*.
3	Создайте *Standard Object (Стандартный объект)* → *Static Text (Статический текст)*. В данном примере используется объект *Static Text5*. Для *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* создайте *процедуру Cи*, которая выводит текст с соответствующим значением тега. Эта *процедура Cи* запускается при изменении тега.
4	Для *Properties (Свойства)* → *Colors (Цвета)* → *Font Color (Цвет шрифта)* создайте *динамический диалог*. В поле *Expression/Formula (Выражение/формула)* укажите тег *S32i_pictu_dyn_00* и выберите запуск по изменению. Выберите *Data Type (Тип данных) Analog (Аналоговый)* и добавьте 4 диапазона величин при помощи кнопки *Add (Добавить)*. Установите диапазоны величин следующим образом: 

Шаг	Процедура: Изменение цвета
5	Для *Properties (Свойства)* → *Flashing(Мигание)* → *Flashing Background Active (Мигающий фон активный)* создайте *динамический диалог*. В поле *Expression/Formula (Выражение/формула)* установите тег *S32i_pictu_dyn_00* и запуск по изменению. Выберите *Data Type (Тип данных) Analog (Аналоговый)* и добавьте диапазон изменения с помощью кнопки *Add (Добавить)*. Установите диапазон изменения следующим образом: 

Процедура Си для статического текста

```
#include "apdefap.h"
 char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{
char text[100];
DWORD temp;

//get tag value
temp = GetTagDWord("S32i_pictu_dyn_00");
//generate text
switch (GetLanguage())
{
case 0x407:
      sprintf(text,"Die Kesseltemperatur beträgt\r\n%d Grad",temp);
      return text;
case 0x409:
      sprintf(text,"Container Temperature is\r\n%d degree",temp);
      return text;
case 0x40C:
      sprintf(text,"La température de chaudière est\r\nde %d degré",temp);
      return text;
default:
      sprintf(text,"Container Temperature is\r\n%d degree",temp);
      return text;
```

Считайте значение тега.

С помощью функции *sprintf* создайте строку, состоящую из текстовой части и числовой величины. Это делается в соответствии с установленным в настоящий момент языком среды исполнения.

Возвращаемая величина — сформированный текст.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

В *динамических диалогах* следует скорректировать используемые диапазоны величин и теги.

3.7.2 Изменение текста (example 02)

Постановка задачи

Автоматически в соответствии со значением тега изменять тексты, привязанные к различным объектам. Аналогичным образом необходимо изменять текст подсказки.

Концепция реализации

Для реализации будем использовать объект *Windows Object (Объект Windows)* → *Button (Кнопка)*, который включает и выключает клапан. *Standard Object (Стандартный объект)* → *Static Text (Статический текст)* используется для отображения включенного или выключенного состояния клапана.

Реализация в проекте WinCC

Шаг	Процедура: Изменение текста
1	В *менеджере тегов* создайте тег типа *Binary Tag (Двоичный тег)*. В данном примере используется тег *BINi_pictu_dyn_01*.
2	Создайте объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В пример используется объект *Button1*. Для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте *процедуру Cи*, которая инвертирует тега *BINi_pictu_dyn_01*.
3	Для *Properties (Свойства)* → *Miscellaneous (Разное)* → *Tooltip Text (Текст подсказки)* создайте *динамический диалог*. В поле *Expression/Formula (Выражение/формула)* укажите тег *BINi_pictu_dyn_01* и выберите срабатывание по изменению. Выберите *Data Type (Тип данных) Bool (Логический)* и в диапазоне значений *Yes/TRUE (Да/истинно)* введите текст *close (закрыт)*, а в диапазоне *No/FALSE (Нет/ложно)* введите текст *open (открыт)*.
4	Создайте *Standard Object (Стандартный объект)* → *Static Text (Статический текст)*. В данном примере используется объект *Static Text7*. Для *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* создайте *динамический диалог*. В поле *Expression/Formula (Выражение/формула)* укажите тег *BINi_pictu_dyn_01* и выберите срабатывание по изменению. Выберите *Data Type (Тип данных) Bool (Логический)* и в диапазоне *Yes/TRUE (Да/истинно)* введите текст *valve opened (клапан открыт)* и в диапазоне *No/FALSE (нет/ложно)* введите текст *valve closed (клапан закрыт)*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

В *динамических диалогах* необходимо скорректировать используемые диапазоны значений и теги.

3.7.3 Анимация движения (example 03)

Постановка задачи

Перемещать объект в определенную позицию на экране в зависимости от значения тега.

Концепция реализации

Для реализации будем использовать объект *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, положение которого управляется тегом. Объект *Windows Object (Объект Windows)* → *Slider Object (Бегунок)* используется для изменения значения тега.

Реализация в проекте WinCC

Шаг	Процедура: Анимация движения
1	В *менеджере тегов* создайте тег типа *Signed 32–Bit Value (32–битная величина со знаком)*. В данном примере используется тег *S32i_pictu_dyn_03*.
2	Создайте *Windows (Объект Windows)* → *Slider Object (Бегунок)*. В данном примере используется объект *Slider Object2*. В диалоге конфигурирования установите *Maximum Value (Максимальная величина)* – *300* и *Minimum Value (Минимальная величина)* – *0*. Для *Events (События)* → *Property Topics (Разделы свойств)* → *Miscellaneous (Разное)* → *Process Driver Connection (Соединение с драйвером процесса)* создайте *прямое соединение* с тегом *S32i_pictu_dyn_03*.
3	Создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В данном примере используется объект *Picture Window1*. Установите *Properties (Свойства)* → *Miscellaneous (Разное)* *Border(Рамка)* и *Adapt Picture(Подгонять размер)* в *Yes (Да)*. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* укажите кадр *pictu_3_chapter_00.pdl*.
4	Для *Properties (Свойства)* → *Geometry (Геометрия)* → *Position X (Координата X)* создайте *динамический диалог*. В поле *Expression/Formula (Выражение/формула)* введите выражение *((S32i_pictu_dyn_03*2)+90)*. Установите срабатывание по изменению тега *S32i_pictu_dyn_03*. Выберите *Data Type (Тип данных) Direct (Непосредственный)*.
5	Для *Properties (Свойства)* → *Geometry (Геометрия)* → *Position Y (Координата Y)* создайте *Динамический диалог*. В поле *Expression/Formula (Выражение/формула)* введите выражение *(400–S32i_pictu_dyn_03)*. Установите срабатывание по изменению тега *S32i_pictu_dyn_03*. Выберите *Data Type (Тип данных) Direct (Непосредственный)*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

В *динамических диалогах* в соответствии с вашими требованиями следует скорректировать выражения для вычисления номера битах.

Также необходимо изменить имя тега.

3.7.4 Отображение и скрытие объектов с использованием побитного опроса (example 04)

Постановка задачи

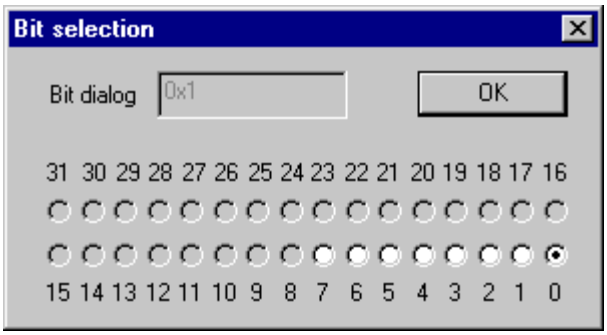
Отображать и скрывать объекты в зависимости от положения определенного бита в теге.

Концепция реализации

Для реализации используется *Windows Object (Объект Windows)* → *Check–Box (Набор флажков)*, который устанавливает отдельные биты в теге. В зависимости от этих битов некоторое количество объектов *Standard Objects (Стандартные объекты)* → *Polygons (Многоугольники)* отображается или скрывается.

Реализация в проекте WinCC

Шаг	Процедура: Отображение и сокрытие объектов с использованием побитного опроса
1	В *менеджере тегов* создайте тег типа *Unsigned 8–Bit Value (8–битная величина без знака)*. В данном примере используется тег U08_pictu_dyn_02.
2	Создайте *Windows Object (Объект Windows)* → *Check–Box (Набор флажков)*. В данном примере используется объект *Check–Box1*. В поле *Properties (Свойства)* → *Geometry (Геометрия)* → *Number of Boxes (Число флажков)* установите количество флажков; в примере — *7*. В поле *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* введите имя объекта, который должен переключаться битом, соответствующим каждому значению индекса. Для → *Events (События)* → *Property Topics (Разделы свойств)* → *Output/Input (Вывод/ввод)* → *Selected Boxes (Выбранные флажки)* → *Change (Изменение)* создайте прямое соединение *источника Property (Свойство)* → *Check–Box 1* → *Selected Boxes (Выбранные флажки)* с *приемником Variable U08i_pictu_dyn_02*.
3	Создайте несколько объектов *Standard Objects (Стандартные объекты)* → *Polygon (Многоугольник)*. В данном примере используются объекты *Polygon1… Polygon7*.

Шаг	Процедура: Отображение и сокрытие объектов с использованием побитного опроса
4	У объекта *Polygon1* создайте *динамический диалог* для *Properties (Свойства)* → *Miscellaneous (Разное)* → *Display (Отображение)*. В поле *Expression/Formula (Выражение/формула)* укажите тег *U08i_pictu_dyn_02* и включите срабатывание по изменению. Выберите *Data Type (Тип данных) Bit (Битовый)*. Используйте кнопку ... для того чтобы открыть диалог выбора битов и выберите первый бит. 
5	Проделайте аналогичные действия с оставшимися объектами *Polygon*, но для каждого из них меняйте номер бита.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

В *динамических диалогах* в соответствии с вашими требованиями следует скорректировать имена тегов и положение кадра.

3.7.5 Анимация движения с использованием процедуры Си (example 05)

Приведенные ниже примеры главы Добавление динамики доступны в проекте *Project_CreatePicture* по щелчку (мыши) на кнопке, изображенной выше. Примеры приведены в кадре *pictu_3_chapter_06a.pdl*.

Постановка задачи

Перемещать объект в одном направлении при щелчке по одной *кнопке* и в другом направлении при щелчке по другой.

Концепция реализации

Для реализации будем использовать *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)* для отображения двух кадров. Два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)* используются для перемещения этого индикатора в двух разных направлениях.

Реализация в проекте WinCC

Шаг	Процедура: Анимация движения с использованием процедуры Си
1	В *менеджере тегов* создайте три тега типа *Binary Tag (Двоичный тег)*, в примере используются теги BINi_pictu_dyn_05, BINi_pictu_dyn_06 и BINi_pictu_dyn_07.
2	Создайте *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*. В данном примере используется объект *Status Display1*. В диалоге конфигурирования укажите тег *BINi_pictu_dyn_05* и установите срабатывание по изменению. Добавьте другое состояние. Для состояния *0* установите изображение *Ferrari1.gif* и для состояния *1* установите изображение *Ferrari2.gif*. 
3	В поле *Properties (Свойства)* → *State (Состояние)* → *Basic Picture Transparent Color (Прозрачный цвет основного рисунка)* установите цвет *White (Белый)* для обоих состояний (*1* и *0)* и установите *Picture Transparent Color On (Включить прозрачный цвет рисунка)* в *Yes (Да)*. Это означает, что белый фон рисунка не отображается..
4	Создайте объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется *Button1*. Для *Events (События)* → *Mouse (Мышь)* → *Press left (Нажатие левой кнопки)* создайте *прямое соединение*, устанавливающее тег *BINi_pictu_dyn_07* в *1* и для *Events (События)* → *Mouse (Мышь)* → *Press right (Нажатие правой кнопки)* создайте *прямое соединение*, сбрасывающее этот тег в *0*.

Шаг	Процедура: Анимация движения с использованием процедуры Си
5	Для второго объекта *Windows Object (Объект Windows)* → *Button (Кнопка)* создайте два *прямых соединения* с тегом *Variable BINi_pictu_dyn_06* аналогично тому, как показано выше. В данном примере используется объект *Button2*.
6	У объекта *Status Display1* создайте *процедуру Си* для *Properties (Свойства)* → *Geometry (Геометрия)* → *Position X (Координата X)*, которая выполняет анимацию движения в зависимости от того, какая кнопка нажата. Установите срабатывание этой процедуры на *250 мс*.

Процедура Си для анимации движения

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN

{
static int a = 90;

//forward
if (GetTagBit("BINi_pictu_dyn_07")&&(a<652)) {
        a+=20;
        SetTagBit("BINi_pictu_dyn_05",(SHORT)!GetTagBit("BINi_pictu_dyn_05"))
        }
//rewind
if (GetTagBit("BINi_pictu_dyn_06")&&(a>-0)) {
        a-=10;
        SetTagBit("BINi_pictu_dyn_05",(SHORT)!GetTagBit("BINi_pictu_dyn_05"))
        }
//return x-position
return a;
}
```

Опишите тег типа *static int* и инициализируйте его текущей координатой X объекта.

Проверьте, какая кнопка нажата и не меньше ли координата X, чем 652. Если это так, то увеличьте переменную, содержащую координату X, на 20. Затем измените рисунок, отображаемый в *Status Display1*.

Проверьте, нажата ли *Button2* и не больше ли координата X чем –200. Если это так, то уменьшите переменную, содержащую координату X, на 10. Затем измените рисунок, отображаемый в *Status Display1*.

Возвращаемая величина — новая координата X.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Принцип анимации может быть перенесен без существенных изменений.

3.7.6 Создание анимации движения с помощью мастера (example 06)

Постановка задачи

Изменять положение объекта на экране при изменениях значения тега. Для координат X и Y должны использоваться отдельные теги. Конфигурирование выполняется при помощи *динамического мастера*.

Концепция реализации

Для реализации будем использовать *Standard Object (Стандартный объект)* → *Circle (Круг)*, который должен перемещаться по экрану. Для ввода тега используются *Windows Objects (Объекты Windows)* → *Slider Objects (Бегунки)*.

Реализация в проекте WinCC

Шаг	Создание анимации движения с помощью мастера
1	В *менеджере тегов* создайте два тега типа *Unsigned 32–Bit Value (32–битная величина без знака)*. В данном примере используются теги *S32i_pictu_dyn_10* и *S32i_pictu_dyn_11*.
2	Создайте два объекта *Windows Objects (Объекты Windows)* → *Slider Objects (Бегунки)*. В данном примере используются *Slider Object1* и *Slider Object2*. Для *Slider Object1* создайте *прямое соединение*. Соедините *источник Properties (Свойства)* → *Slider Object1* → *Process Driver Connection (Соединение с драйвером процесса)* с переменной *S32i_pictu_dyn_10*. Аналогичным образом создайте *прямое соединение* с переменной *S32i_pictu_dyn_11* для *Slider Object2*.
3	В *диалогах создания бегунков* установите *Maximum Value (Максимальная величина) –255*.
4	Создайте объект *Standard Object (Стандартный объект)* → *Circle (Круг)*. В данном примере используется объект *Circle1*. Пока объект выделен, выберите вкладку *Standard Dynamics (Стандартная динамика)* и затем пункт *Move Object (Движение объекта) динамического мастера* с помощью D (двойного щелчка мыши). В качестве триггера выберите *Tag (Тег)*. На странице *Set Options (Установка опций)* выберите тег *S32i_pictu_dyn_10* для координаты X и *S32i_pictu_dyn_11* для координаты Y. Введите *0* и *255* соответственно в качестве нижней и верхней границ форматирования. На следующей странице определите участок кадра, в пределах которого должен перемещаться объект. Щелкните по кнопке *Finish (Завершение)* для завершения мастера.
5	В *процедурах Cи*, созданных *динамическим мастером*, для тегов, используемого в *Properties (Свойства)* → *Geometry (Геометрия)* → *Position X (Координата X)* и в *Properties (Свойства)* → *Geometry (Геометрия)* → *Position Y (Координата Y)*, установите срабатывание по изменению.

Процедура Си, созданная мастером для координаты X

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN

{
long i,j,k;
i=GetTagWord("S32i_pictu_dyn_10");
j=((i-0)*100/(255-0));
k=min((((j*(690-490))/100)+490),690);
return max(490,k);

}
```

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Настройки, сделанные в *динамическом мастере* для анимации движения, следует привести в соответствие с вашими требованиями.

3.7.7 Изменение цвета с помощью процедуры Си (example 06)

Постановка задачи

Плавно изменять цвет объекта от темного к светлому по мере изменения значения тега.

Концепция реализации

Для реализации будем использовать объект *Standard Object (Стандартный объект)* → *Circle (Круг)*, цвет которого меняется вместе со значением тега. Для ввода значения тега используем *Windows Object (Объект Windows)* → *Slider Object (Бегунок)*.

Реализация в проекте WinCC

Шаг	Процедура: Изменение цвета с помощью процедуры Си
1	В *менеджере тегов* создайте тег типа *Unsigned 32–Bit Value (32–битная величина без знака)*. В данном примере используется тег *S32i_pictu_dyn_10*.
2	Создайте *Windows Object (Объект Windows)* → *Slider Object (Бегунок)*. В данном примере используется объект *Slider Object1*. Для *Events (События)* → *Property Topics (Разделы свойств)* → *Miscellaneous (Разное)* → *Process Driver Connection (Связь с драйвером процесса) Slider Object1*, создайте *прямое соединение*. Соедините *источник Properties (Свойства)* → *Slider Object1* → *Process Driver Connection (Связь с драйвером процесса)* с переменной *S32i_pictu_dyn_10*.
3	В *Slider Object1* установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Maximum Value (Максимальная величина) – 255*.
4	Создайте *Standard Object (Стандартный объект)* → *Circle (Круг)*, в примере используется объект *Circle1*. Для *Properties (Свойства)* → *Colors (Цвета)* → *Background Color (Цвет фона)* создайте *процедуру Си*, изменяющую цвет в соответствии со значением тега *S32i_pictu_dyn_10*. Эта процедура запускается по изменению этого тега.

Процедура Си для изменения цвета

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN

{
return (GetTagDWord("S32i_pictu_dyn_10")<<8);
}
```

Процедура возвращает тег *S32i_pictu_dyn_10*, сдвинутый влево на 8 разрядов.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Цвета кодируются указанием значений для 3 основных составляющих: красного, зеленого и синего. В 24–разрядном коде для представления цвета под каждую составляющую отводится восемь бит. В данном примере значение тега сдвигается на 8 разрядов влево и, следовательно, описывает зеленую составляющую. Если сдвиг не производить, цвет будет меняться от черного к красному; если тег сдвигать на 16 разрядов, то от черного к синему.

3.7.8 Анимация движения с использованием индикатора состояния (example 07)

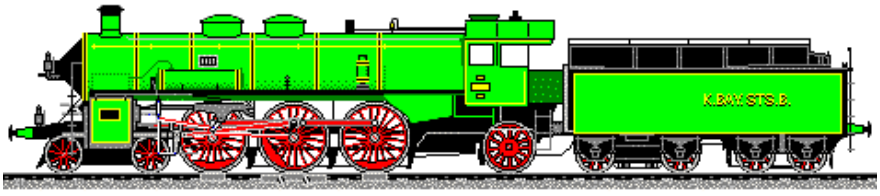
Постановка задачи

Имитировать движение посредством чередования разных рисунков в *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*.

Концепция реализации

Для реализации будем использовать *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*, в котором при помощи еще одного объекта *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)* будем чередовать рисунки после включения индикатора

Реализация в проекте WinCC

Шаг	Процедура: Анимация движения с использованием индикатора состояния
1	В *менеджере тегов* создайте тег типа *Binary Tag (Двоичный тег)*. В данном примере используется тег *BINi_pictu_dyn_09*.
2	Создайте *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*. В данном примере используется объект *Status Display3*. В *диалоге конфигурирования* укажите тег *BINi_pictu_dyn_09* и срабатывание по изменению. Добавьте новое *состояние*. Для *status 0* установите рисунок *Smili.gif* и для *status 1* — рисунок *Ohh.gif*.
3	У объекта *Status Display3* создайте *процедуру Си* для *Events (События)* → *Mouse (мышь)* → *Press Left (Нажатие левой кнопки)*, которая инвертирует тег *BINi_pictu_dyn_09*.
4	Создайте другой *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*, в примере используется объект *Status Display4*. В поле *Properties (Свойства)* → *State (Состояние)* → *Current Status (Текущее состояние)* добавьте семь дополнительных состояний с соответствующими рисунками. Для каждого состояния в атрибуте *Basic Picture Transparent Color (Прозрачный цвет основного рисунка)* выберите *White (Белый)* и в *Basic Picture Transparent Color On (Включить прозрачный цвет основного рисунка)* — *Yes (Да)*. Каждому из состояний от *0* до *7* присваивается один из рисунков от *S3_61.gif* до *S3_68.gif*. 

Шаг	Процедура: Анимация движения с использованием индикатора состояния
5	У объекта *Status Display4* для *Properties (Свойства)* → *State (Состояние)* → *Current Status (Текущее состояние)* создайте *процедуру Си*, которая инициирует изменение текущего состояния от 0 to 7. Установите срабатывание этой процедуры каждые *250 мс*.

Процедура Си для Status Display4

```
#include "apdefap.h"
long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertyN

{
static int a = 0, b = 0;

if (GetTagBit("BINi_pictu_dyn_09")) {
        if (b==0) a++;
        else a--;
        if (a==7) b=1;
        if (a==0) b=0;
        }
return a;
}
```

Объявите два тега типа *static int* и инициализируйте их нулем.

Если анимация запущена, двигайтесь по тегам от 0 до 7 и затем вновь с нуля.

Возвращайте этот тег.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Принцип анимации может быть заимствован без существенных изменений.

Объект *Status Display3* может использоваться в других проектах в качестве переключаемого объекта при соответствующей коррекции рисунков состояний и имен тегов.

3.8 Переключение языка

Примеры, имеющие отношение к этой теме, доступны в проекте *Project_CreatePicture* по нажатию (мышью) на *кнопке*, изображенной выше. Примеры приведены в кадре *pictu_3_chapter_07.pdl*.

3.8.1 Переключение языка режима исполнения (example 01)

Постановка задачи

Язык режима исполнения должен переключаться посредством одной кнопки для каждого языка.

Концепция реализации

Для реализации будем использовать три объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*, которые можно взять полностью готовыми из библиотеки.

Реализация в проекте WinCC

Шаг	Процедура: Переключение языка режима исполнения
1	В *графическом дизайнере* создайте любой кадр, используя нужный язык. С помощью команды меню *View (Вид)* → *Language (Язык)* выбирается другой язык, и все тексты переводятся на этот язык. Используя программу *language.exe*, размещенную на компакт–диске WinCC, все используемые в проекте тексты, могут быть экспортированы в файл *csv*. После этого их можно перевести и вновь импортировать в проект.
2	Откройте библиотеку, используя команду меню *View (Вид)* → *Library (Библиотека)*. Из папки *Global Library (Глобальная библиотека)* → *Buttons Language (Язык кнопок)* выберите соответствующие *Buttons (Кнопки)*. Проще всего это сделать, щелкнув по желаемому объекту и перетащив его в рабочее поле, удерживая нажатой кнопку (мыши).
3	Если требуется язык, не входящий в библиотеку, у объекта *Windows Object (Объект Windows)* → *Button (Кнопка)* необходимо создать *процедуру Си* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*, которая выполняет соответствующее переключение языка. Кроме того, для создания соответствующей *процедуры Си* можно использовать *динамический мастер*. Для использования мастера выделите *кнопку*, перейдите на вкладку *System Functions (Системные функции)* и затем с помощью D (двойного щелчка мыши) выберите пункт *Language Switch (Переключение языка) динамического мастера*. В этом мастере можно выбрать требуемый язык.

Процедура Си для кнопки на немецком языке

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF

{
SetLanguage(0x407);    //Rückgabe-Typ :BOOL
}
```

Используйте функцию *SetLanguage* для смены настроенного языка, введя соответствующий код.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

В *динамическом мастере* произведите необходимые языковые установки.

3.8.2 Диалоговая панель для переключения языка режима исполнения и среды WinCC (example 02)

Постановка задачи

По нажатию *кнопки* вызывать диалоговую панель, в которой можно выбрать один из языков.

Концепция реализации

Для реализации будем использовать объект *Windows Object (Объект Windows)* → *Button (Кнопка)*, который отображает или скрывает *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. Диалоговая панель можно позаимствовать непосредственно из проекта *Project_CreatePicture*.

Реализация в проекте WinCC

Шаг	Процедура: Диалоговая панель для переключения языка режима исполнения и среды WinCC
1	В *графическом дизайнере* создайте любой кадр, используя нужный язык. С помощью команды меню *View (Вид)* → *Language (Язык)* выбирается другой язык, и все тексты переводятся на этот язык.
2	Создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В данном примере используется объект *Picture Window1*. В поле *Properties (Свойства)* → *Geometry(Геометрия)* установите *Window Width (Ширина окна) – 230* и *Window Height (Высота окна) – 214*. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* установите пункты *Moveable (Перемещаемый)*, *Border (Рамка)*, *Title (Заголовок)* и *Can Be Closed (Может быть закрыто) – Yes(Да)*. В пункте *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* укажите кадр *pictu_5_window_19.pdl*. Этот кадр размещается в проекте *Project_CreatePicture* и может быть использован без изменений. Установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Display (Отображение)* в *No (Нет)*.
3	Создайте объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере используется объект *Button4*. Для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте *прямое соединение*, которое делает объект *Picture Window1* видимым.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Кадр *pictu_5_window_19.pdl* может быть повторно использован в другом проекте без изменений.

3.9 Работа без мыши

Примеры, имеющие отношение к этой теме, доступны в проекте *Project_CreatePicture* по нажатию (мышью) на *кнопке*, изображенной выше. Примеры приведены в кадре *pictu_3_chapter_08a.pdl*.

3.9.1 Работа с помощью клавиши TAB или горячих клавиш (example 01)

Постановка задачи

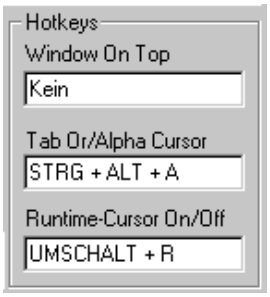
Произвести форматирование текста с помощью различных диалогов.
Необходимо установить цвет и различные свойства шрифта, например, размер.
Дополнительно следует иметь возможность возвращать сделанные установки к значениям по умолчанию.
Управление всеми элементами кадра должно выполняться исключительно при помощи клавиатуры.

Концепция реализации

Для реализации будем использовать четыре объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. Они делают диалоги видимыми. Диалоги должны управляться клавиатурой, если включен курсор режима исполнения. Выбор кнопки осуществляется клавишей TAB. Кроме того, каждой кнопке назначается горячая клавиша. Для отображения диалогов используются три *Smart Objects (Интеллектуальные объекты)* → *Picture Windows (Окна кадра)*.

Конфигурирование управления курсором

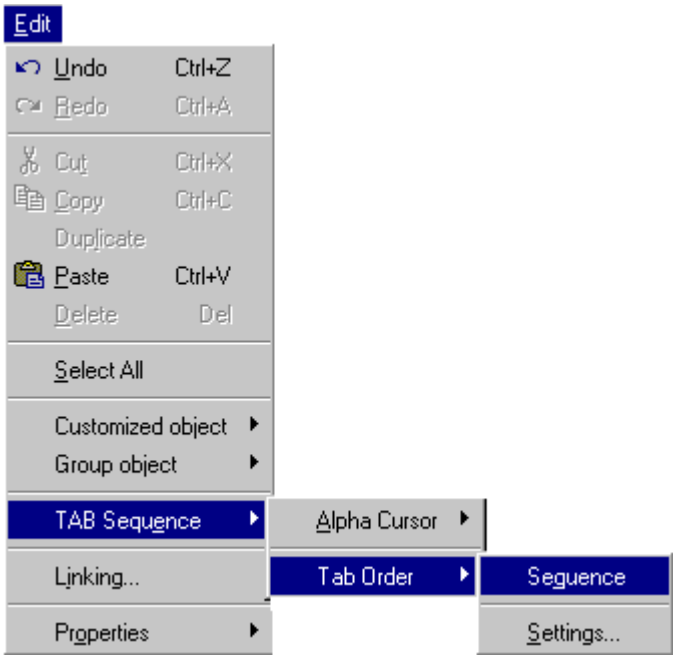
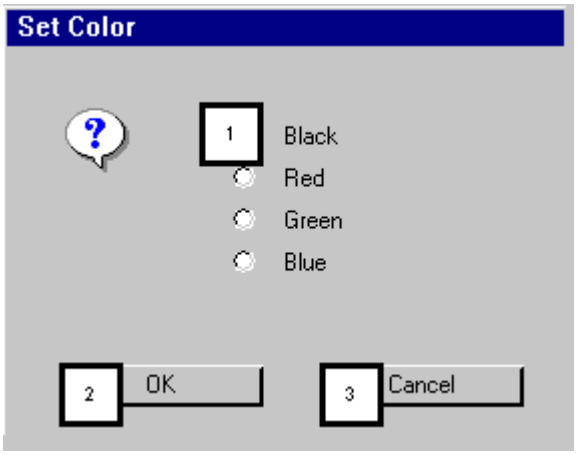
Шаг	Процедура: Конфигурирование управления курсором
1	В *Control Center (Центр управления)* произведите настройку управления курсором. Щелкните R (правой кнопкой мыши) на пункте *Computer (Компьютер)* и затем выберите пункт *Properties (Свойства)* из выпадающего меню. В следующем диалоге *Computer List Properties (Свойства списка компьютеров)* щелкните (мышью) по кнопке *Properties (Свойства)*.  Выберите вкладку *Graphics Runtime (Графика режима исполнения)*.

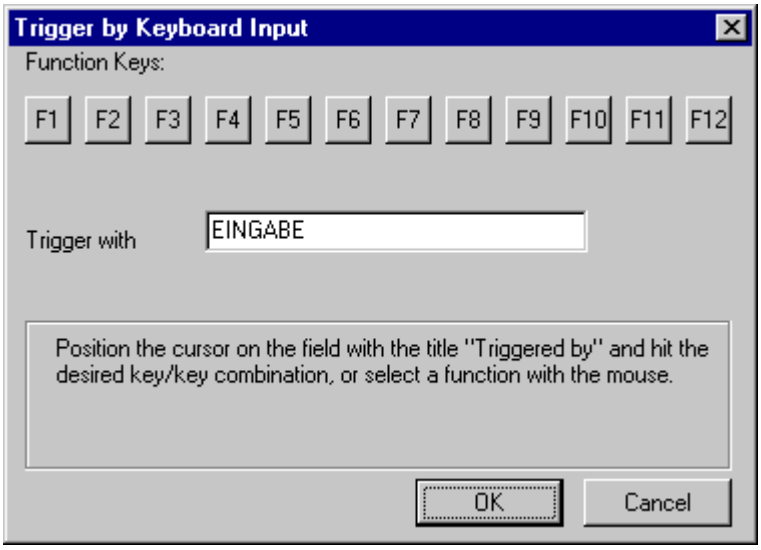
Шаг	Процедура: Конфигурирование управления курсором
2	Настройте горячие клавиши следующим образом. Для команды *Window on Top (Окно на переднем плане)* не создается горячей клавиши, так как в примерах рабочий фокус устанавливается посредством *процедур Cu*. Для переключения между *Tab Order/Alpha Cursor (Порядок переключения по клавише Tab/Альфа–курсор)* установите *SHIFT+A*, и для включения/выключения опции *Runtime Cursors (Курсоры режима исполнения)* установите *SHIFT+R*. 
3	В поле *Cursor Control: Keys (Управление курсором: клавиши)* никаких клавиш указывать не требуется. Если они нужны в примерах, они будут заданы с использованием функции API. Это делается для демонстрации различных вариантов реализации. В обычных случаях для проекта выбирается и устанавливается один конкретный вариант.

Реализация в проекте WinCC

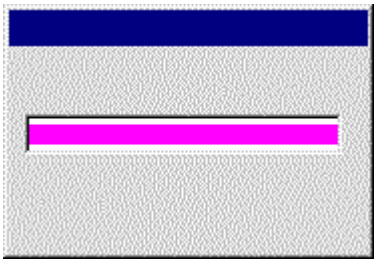
Шаг	Процедура: Конфигурирование управления курсором
1	В *менеджере тегов* создайте три тега типа *Unsigned 16–Bit Value(16–битная величина без знака)*, которые будут содержать установленные свойства шрифта. В данном примере используются теги от *U16i_pictu_cursor_00* до *U16i_pictu_cursor_02*.
2	В кадре *pictu_3_chapter_08.pdl* создайте четыре объекта типа *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере это объекты *Button1*, *Button2*, *Button3* и *Button4*. Они используются для отображения диалогов и отказа от сделанных установок. Кроме того, создайте *Standard Object (Стандартный объект)* → *Static Text (Статический текст)*, свойства шрифта которого устанавливаются в диалогах. В данном примере используется объект *Static Text1*.

Шаг	Процедура: Конфигурирование управления курсором
3	Создайте другой кадр, который служит диалогом установки цвета. В данном примере это кадр *pictu_5_window_23.pdl*. Создайте в этом кадре *Windows Object(Объект Windows)* → *Option Group (Группа опций)*. В данном примере это объект *Option Group1*. Установите *Property (Свойства)* → *Geometry (Геометрия)* → *Number of Boxes (Число пунктов) – 4*. Таким образом разрешается выбор четырех разных цветов. Для *Properties (Свойства)* → *Output/Input (Вывод/Ввод)* → *Selected Boxes (Выбранные пункты)* создайте *соединение с тегом U16i_pictu_cursor_00*. Для *Properties (Свойства)* → *Geometry (Геометрия)* → *Position X (Координата X)* создайте *процедуру Cи*, которая устанавливает фокус на созданный объект. Эта *процедура Cи* срабатывает один раз за *1 h (1 час)*. Если *группа опций* находится в рабочем фокусе при активном курсоре режима исполнения, она отображается с рамкой. Если прямоугольники цвета фона размещаются поверх рамки, рамка объекта может быть сделана невидимой.
4	Создайте в том же кадре два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. В примере это объекты *Button1* и *Button2*. *Button1* используется как кнопка *OK*. Для *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* создайте *процедуру Cи*, которая меняет цвет текста в зависимости от значения тега *U16i_pictu_cursor_00* и затем скрывает этот диалог. *Button2* используется как кнопка *Cancel (Отмена)*. В *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* создайте *прямое соединение*, которое делает окно невидимым.

Шаг	Процедура: Конфигурирование управления курсором
5	Сделайте установки для работы клавиатуры. Установите порядок выбора объектов клавишей *Tab*. Это делается с помощью команды меню *Edit (редактирование)* → *TAB Sequence (Последовательность TAB)* → *Tab Order (Порядок Tab)* → *Sequence (Последовательность)*.  Каждый объект, которым можно управлять, теперь отображается с номером. Порядок номеров представляет порядок обхода по клавише TAB. Порядок можно изменить, щелкая мышью по индивидуальным номерам. Порядок устанавливается следующим образом: 

Шаг	Процедура: Конфигурирование управления курсором
	Выбор в *группе опций* осуществляется при помощи *клавиш курсора*. Выбор цвета осуществляется клавишей "пробел". Клавиша *TAB* используется для переключения между элементами управления. Кнопки управляются клавишей "пробел". Кроме того, обеим *кнопкам* назначаются горячие клавиши. Для *Properties (Свойства)* → *Miscellaneous (Разное)* → *Hot Key (Горячая клавиша)* открывается диалог конфигурации горячих клавиш. Для кнопки *OK* устанавливается клавиша *ENTER*, для кнопки *Cancel* — клавиша *ESC*. 
6	В кадре *pictu_3_chapter_08.pdl* создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, в котором отображается созданный кадр. В данном примере это объект *Picture Window1*. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (имя кадра)* укажите кадр *pictu_3_window_23.pdl*. Установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Display (Отображение) – No (Нет)*.
7	Для *Button1* создайте *процедуру Cu*, которая запрашивает текущий цвет текста и записывает результат в тег *U16i_pictu_cursor_00*. Это сделано для установки текущего значения в *группе выбора* диалога. Кроме этого отображается объект *Picture Window1*. У *Button1* создайте *процедуру Cu* для *Properties (Свойства)* → *Geometry (Геометрия)* → *Position X (Координата X)*, которая устанавливает рабочий фокус на этот объект. Эта *процедура Cu* срабатывает один раз за *1 h (1 час)*, однако фокус устанавливается только первый раз. **Set Color F9** Кроме того, *кнопке* назначается горячая клавиша. В примере это функциональная клавиша *F9*.

Шаг	Процедура: Конфигурирование управления курсором
8	Создайте кадр, который будет служить диалогом для установки различных свойств шрифта. В примере это кадр *pictu_5_window_24.pdl*. Создайте в этом кадре *Windows Object (Объект Windows)* → *Check–Box (Группа флажков)*. В данном примере используется объект *Check–Box1*. Установите *Property (Свойство)* → *Geometry (Геометрия)* → *Number of Boxes (Число флажков)* – *4*. Необходимо обеспечить возможность выбора свойств *Bold (Жирный)*, *Italic (Курсив)*, *Underline (Подчеркнутый)* и *Border (Рамка)*. Для *Properties (Свойства)* → *Output/Input (Вывод/ввод)* → *Selected Boxes (Выбранные флажки)* создайте *соединение с тегом U16i_pictu_cursor_01*. Для *Properties (Свойства)* → *Geometry (Геометрия)* → *Position X (Координата X)* создайте *процедуру Cu*, которая переносит фокус на этот объект. Рамка выбора может быть скрыта при помощи процедуры, описанной для *группы выбора*. Так же как и для кадра *pictu_5_window_23.pdl* создайте два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. При нажатии кнопки *OK* считывается тег *U16i_pictu_cursor_01*, и к тексту применяются соответствующие установки. Установки, относящиеся к работе с клавиатурой, делаются аналогично тому, как это было сделано для кадра *pictu_5_window_23.pdl*.
9	В кадре *pictu_3_chapter_08.pdl* создайте другое *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, в котором отображается созданный кадр. В данном примере это объект *Picture Window2*. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* укажите кадр *pictu_3_window_24.pdl*. Установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Display (Отображение)* в *No (Нет)*.
10	Для *Button2* создайте *процедуру Cu*, которая запрашивает текущие свойства шрифта и в зависимости от результата записывает их в тег *U16i_pictu_cursor_01*. Это сделано для того, чтобы отметить выбранные пункты *группе выбора* диалога. Помимо этого отображается объект *Picture Window2*. Далее этой *кнопке* назначается горячая клавиша. В примере это функциональная клавиша *F10*. **Format F10**

Шаг	Процедура: Конфигурирование управления курсором
11	Создайте кадр, который будет служить диалогом для установки размера шрифта. В примере это кадр *pictu_5_window_25.pdl*. В этом кадре создайте *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*. В данном примере используется объект *I/O Field1*. Для *Properties (Свойства)* → *Output/Input(Вывод/ввод)* → *Output Value (Выводимое значение)* создайте *соединение с тегом U16i_pictu_cursor_02*. Для *Properties (Свойства)* → *Geometry (Геометрия)* → *Position X (Координата X)* создайте *процедуру Cu*, которая устанавливает фокус на этот объект. Рамка выбора может быть скрыта путем размещения *графического объекта* поверх *поля ввода/вывода*. В данном примере используется объект *Graphic Object1*. Рисунок, отображаемый *графическим объектом*, имеет тот же цвет, что и область отображения *поля ввода/вывода*. Этот цвет *графического объекта* задается в поле *Properties (Свойства)* → *Picture (Кадр)* → *Picture Transparent Color (Прозрачный цвет рисунка)*. Кроме того, атрибут *Property (Свойство)* → *Picture (Кадр)* → *Picture Transparent Color On (Включение прозрачного цвета кадра)* устанавливается в *Yes (Да)*. Используемый рисунок отображается следующим образом. 
12	Аналогично кадру *pictu_5_window_23.pdl* создайте два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. По нажатию кнопок *OK* или *Cancel* диалог закрывается. Однако обе *кнопки* исключены из порядка обхода по клавише TAB. Это делается с помощью команды меню *Edit (Правка)* → *TAB Sequence (Последовательность Tab)* → *Tab Order (Порядок Tab)* → *Sequence (Последовательность)*. Объект может быть удален из порядка обхода по клавише TAB путем нажатия и удержания комбинации клавиш *CTRL+SHIFT* и выбора его . Вместо номера в белом прямоугольнике отображается "*".  Управление *кнопками* осуществляется исключительно с помощью горячих клавиш *ENTER* и *ESC*. Если нажата *ENTER*, величина, введенная в *поле ввода/вывода*, переносится в тег *U16i_pictu_cursor_02*. Для объекта *Static Text1* кадра *pictu_3_chapter_08.pdl* в поле *Properties (Свойства)* → *Font (Шрифт)* → *Font Size (Размер шрифта)* создайте *соединение с тегом U16i_pictu_cursor_02*.

Шаг	Процедура: Конфигурирование управления курсором
13	В кадре *pictu_3_chapter_08.pdl* создайте другое *Smart Object(Интеллектуальный объект)* → *Picture Window (Окно кадра)*, в котором отображается созданный кадр. В данном примере это объект *Picture Window3*. В *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* укажите кадр *pictu_3_window_25.pdl*. Установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Display (Отображение) – No (Нет)*.
14	Создайте для *Button3 процедуру Си*, которая отображает объект *Picture Window3*. Помимо этого *кнопке* назначается горячая клавиша. В примере это функциональная клавиша *F11*. **Font Size F11**
15	Создайте для *Button4 процедуру Си*, которая устанавливает изменяемые свойства объекта *Static Text1* в исходное состояние. *Кнопке* назначается горячая клавиша *F12*. **Reset F12**
16	С помощью команды меню *Edit (Правка)* → *TAB Sequence (Последовательность Tab)* → *Tab Order (Порядок Tab)* → *Sequence (Последовательность)* объекты от *Button1* до *Button4* устанавливаются в соответствующем порядке. Все другие объекты удаляются из очередности TAB. *Кнопки* подтверждаются нажатием *"Пробела"*, или соответствующей горячей клавиши.

Процедура Си для установки фокуса

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropert

{
Static BOOL  bFirst = FALSE;

//set focus in first run
if (bFirst==FALSE)
  Set_Focus(lpszPictureName,lpszObjectName);

BFirst=TRUE;
Return 100;
}
```

При первом вызове функции фокус устанавливается на объект–владелец. *Процедура Си* вызывается один раз в течение каждого часа. Фокус, однако, устанавливается только один раз.

Эта *процедура Си* создается для *Property (Свойства)* → *Geometry (Геометрия)* → *Position X (Координата X)* объекта *Option Group1* в кадре *pictu_5_window_25.pdl*. Она выполняется в часовом цикле.

Процедура Си для установки цвета шрифта

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{

long int wValue;

//get text color
wValue=GetForeColor(lpszPictureName,"Static Text1");

//set tag whitch contains text color
switch (wValue)
{
      case CO_BLACK: SetTagWord("U16i_pictu_cursor_00",1);
                     break;
      case CO_RED  : SetTagWord("U16i_pictu_cursor_00",2);
                     break;
      case CO_GREEN: SetTagWord("U16i_pictu_cursor_00",4);
                     break;
      case CO_BLUE : SetTagWord("U16i_pictu_cursor_00",8);
                     break;
}

//open dialoge box
SetVisible(lpszPictureName,"Picture Window1",TRUE);

}
```

Свойство *Font Color (Цвет шрифта)* объекта *Static Text1* устанавливается в зависимости от значения тега *U16i_pictu_cursor_00*.

Эта *процедура Си* выполняется по нажатию кнопки *OK* в кадре *pictu_5_window_23.pdl*.

Процедура Си при открытии кадра

```
#include "apdefap.h"
void OnOpenPicture(char* lpszPictureName, char* lpszObjectName, char* lpszPi

{
//load DLL
#pragma codc ("pdlrtapi.dll")
#include <pdlrtapi.h>
#pragma code ()

PDLRTSetCursorKeys(255,255,255,255,0,0,NULL, (LPVOID)1,NULL);
}
```

Если выбран кадр *pictu_3_chapter_08.pdl*, клавиши курсора устанавливаются функцией API *PDLRTSetCursorKeys*. Первые четыре параметра функции содержат коды желаемых клавиш для движения вверх/вниз и влево/вправо.

В данном примере используется клавиша WIN для всех направлений курсора с тем, чтобы отключить возможность управления курсором стрелками. Курсор режима исполнения может, следовательно, перемещаться только с помощью клавиши TAB в установленном порядке.

Замечание относительно основных применений

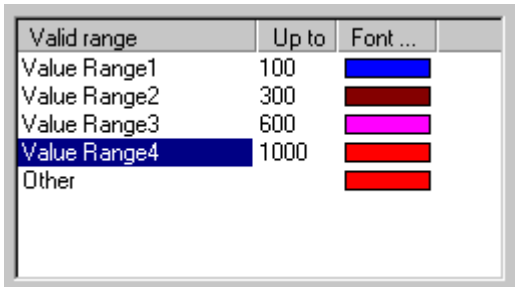
В общем случае перед использованием описанных приемов необходимо учесть следующее:

Если используется несколько окон, комбинации клавиш для переключения между ними должны быть определены в Control Center (Центре управления). В данном примере выбран вариант реализации, не предусматривающий переключение между отдельными диалогами с помощью клавиатуры.

Используемые горячие клавиши и комбинации клавиш должны быть настроены в соответствии с вашими требованиями.

Этот пример построен таким образом, что для перемещения курсора в режиме исполнения не требуется никаких специальных клавиш, кроме TAB. Однако для работы групп опций и флажков используются клавиши курсора, назначенные по умолчанию.

3.9.2 Клавиатура курсора (example 02)



Доступ к примеру в кадре *pictu_3_chapter_08.pdl* можно получить по нажатию комбинации клавиш *CTRL+W* или щелчку (мыши) на изображенной выше кнопке. Он приведен в кадре *pictu_3_chapter_08a.pdl*.

Постановка задачи

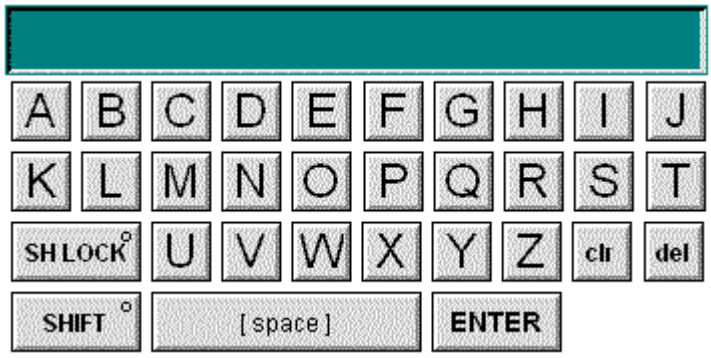
Ввести текст с помощью клавиш курсора и клавиатуры, созданной в кадре. Выбор отдельных символов осуществляется с помощью клавиш курсора. Поведение курсора может быть задано в режиме исполнения с использованием диалога. Диалог отображается только после нажатия горячей клавиши.

Концепция реализации

Для реализации используется готовая клавиатура из библиотеки. Эта клавиатура может быть адаптирована в соответствии с вашими требованиями.
Диалог отображается в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. Для включения отображения диалога используется объект *Windows Object (Объект Windows)* → *Button (Кнопка)*, которому назначается горячая клавиша. Сама кнопка в режиме исполнения не отображается.

Реализация в графическом дизайнере

Шаг	Процедура: Реализация в графическом дизайнере
1	В менеджере тегов создаются два тега типа *Unsigned 16–Bit Value (16–битная величина без знака)*, которые хранят установки поведения курсора. В данном примере используются теги *U16i_pictu_cursor_04* и *U16i_pictu_cursor_05*. В дополнение создается тег типа *Text Tag 16–Bit Character Set (Текстовый тег 16–битного набора символов)*, в который заносится введенный текст. В примере используется тег *T16i_pictu_cursor_00*.

Шаг	Процедура: Реализация в графическом дизайнере
2	Откройте библиотеку с помощью кнопки  панели инструментов. Из папки клавиатур выберите *объект Keyboard Char (Символьная клавиатура)* и перетащите его в кадр. В примере это кадр *pictu_3_chapter_08a.pdl*. Поясняющие объекты можно удалить, требуемые элементы показаны ниже: 
3	У кнопки *ENTER* создайте *процедуру Cи* для *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)*, которая записывает введенный текст в текстовый тег. Имя этого тега *ConnectedVarChar*. Измените его на *T16i_pictu_cursor_00*. В данном примере содержимое тега *T16i_pictu_cursor_00* отображается в статическом тексте с заголовком кадра. Это выполняется с помощью соединения с указанным тегом.
4	Настройте управление курсором. Все объекты, кроме клавиш клавиатуры, удаляются из порядка обхода по клавише TAB. Сам порядок обхода изменять не требуется, так как работа клавиатуры должна осуществляться с помощью клавиш курсора, а не клавиш табуляции. Настройка клавиш управления курсором производятся в *процедуре Cи* для *Events (События)* → *Miscellaneous (Разное)* → *Open Picture (Открытие кадра)*.
5	Поведение курсора может устанавливаться в диалоговой панели. В обычных случаях это уже сделано в *Control Center (Центре управления)*. Щелкните R (правой кнопкой мыши) по элементу *Computer (Компьютер)* и выберите пункт *Properties (Свойства)* выпадающего меню. В следующем диалоге *Computer List Properties (Свойства списка компьютера)* щелкните (мышью) по кнопке *Properties (Свойства)*. Выберите вкладку *Graphics Runtime (Графика режима исполнения)*. В поле *Cursor Control: Keys (Управление курсором: клавиши)* можно выбрать три режима. 

Шаг	Процедура: Реализация в графическом дизайнере
6	Создайте новый кадр для окна диалога. В данном примере это кадр *pictu_5_window_26.pdl*. Создайте в нем три объекта *Smart Objects (Интеллектуальные объекты)* → *Status Displays (Индикаторы состояния)*. В примере это объекты *Status Display1*, *Status Display2* и *Status Display3*. С помощью *диалога конфигурирования* установите рисунки для каждого *индикатора состояния*, отображающие нажатое и отпущенное состояния кнопки. *Status 1* представляет нажатую кнопку, *Status 0* — отпущенную. Для *Properties (Свойства)* → *State(Состояние)* → *Current Status (Текущее состояние)* создайте *динамический диалог*, который управляет текущим состоянием в зависимости от тега *U16i_pictu_cursor_05*. Этот тег содержит временные установки поведения курсора. Для *Events (События)* → *Keyboard (Клавиатура)* → *Press (Нажатие)* создайте *процедуру Cu*, которая записывает значение в тег *U16i_pictu_cursor_05*, представляющий сделанный выбор. Это величины *0...Normal (Нормальный)*, *1...Normal without Rollback (Нормальный без отката)*, *10...Extended (Расширенный)*
7	В том же кадре создайте два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. В примере это объекты *Button1* и *Button2*. *Button1* используется как кнопка *OK*. Для *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* создайте *процедуру Cu*, которая записывает значение тега *U16i_pictu_cursor_05* в тег *U16i_pictu_cursor_04*. Этот тег представляет текущее поведение курсора. После этого функция API *PDLRTSetCursorKeys* переключает поведение курсора. Величина, хранящаяся в теге *U16i_pictu_cursor_04*, соответствует численному значению, ожидаемому функцией для определения поведения курсора. Кроме того, фокус устанавливается на клавишу *A* клавиатуры, и окно становится невидимым. *Button2* используется как кнопка *Cancel (Отмена)*. Для *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* создайте *процедуру Cu*, которая устанавливает фокус на клавишу *A* клавиатуры и прячет окно.
8	В кадре *pictu_3_chapter_08a.pdl* создайте *Smart Object (Интеллектуальный объект)* → *Picture Window (окно кадра)*, в котором отображается созданный кадр. В данном примере это объект *Picture Window1*. В *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* установите кадр *pictu_3_window_26.pdl*. Установите *Property* → *Miscellaneous* → *Display (Отображение) – No(Нет)*.
9	В кадре *pictu_3_chapter_08a.pdl* создайте объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере это объект *Button5*. Для *Button5* создайте *процедуру Cu*, которая записывает код текущего поведения курсора в тег *U16i_pictu_cursor_05* и отображает объект *Picture Window1*. Кроме того, данной *кнопке* назначается горячая клавиша. В примере это комбинация клавиш *CTRL+E*. Установите атрибут *Property (Свойство)* → *Miscellaneous (Различное)* → *Display (Отображение) – No (Нет)*. Таким образом, кнопка скрывается, но горячая клавиша остается активной.

Процедура Си для события нажатия клавиши

```
#include "apdefap.h"
void OnKeyDown(char* lpszPictureName, char* lpszObjectName, char* lpszPrc

{
if (nChar==VK_SPACE)
        SetTagWord("U16i_pictu_cursor_05",0);
}
```

Если курсор режима исполнения находится поверх *индикатора состояния*, эта *процедура Си* будет выполняться при нажатии любой клавиши. Тег *nChar* содержит код соответствующей клавиши. Если это пробел, соответствующий код поведения курсора записывается в тег. В данном примере это код нормального поведения курсора.

Эту *процедуру Си* следует сконфигурировать для события "нажатие клавиши", так как объект не кнопка, а индикатор состояния. В противном случае можно использовать событие "действие мыши".

При конфигурировании объекта для работы с мышью следует создать другую *процедуру Си*, отвечающую за событие "действие мыши", для которого отсутствует запрос кода клавиши.

Процедура Си для кнопки OK

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert

{
//load DLL
#pragma code ("pdlrtapi.dll")
#include <pdlrtapi.h>
#pragma code ()

//set selected cursor mode to tag
SetTagWord("U16i_pictu_cursor_04",
        GetTagWord("U16i_pictu_cursor_05"));

//set cursor mode
PDLRTSetCursorKeys(VK_UP,VK_DOWN,VK_LEFT,VK_RIGHT,0,
        GetTagWord("U16i_pictu_cursor_04"),NULL,
        (LPVOID)1,NULL);

//set focus to A-button
Set_Focus("pictu_3_chapter_08a.PDL","Button92");

//close window
SetVisible("pictu_3_chapter_08a.PDL","Picture Window1" ,FALSE);
}
```

Загрузка DLL, содержащей функцию *PDLRTSetCursorKeys*.

Поведение курсора сохраняется в теге *U16i_pictu_cursor_04*.

Установка курсора осуществляется с помощью функции API *PDLRTSetCursorKeys*. Первые четыре параметра функции содержат коды нужных клавиш для движения вверх/вниз и влево/вправо. Шестой параметр используется для передачи функции желаемого поведения курсора. Этот параметр уже содержится (в корректной кодировке) в теге *U16i_pictu_cursor_04*.

Фокус устанавливается на клавишу *A* на клавиатуре и диалог закрывается.

Для события "Открытие кадра" также вызывается функция *PDLRTSetCursorKeys,* и поведение курсора приводится в нормальное состояние. В это время DLL уже загружена. Загружать ее повторно не нужно. Однако, ради завершенности, об этом упоминается еще раз.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Используемые клавиатурные комбинации и горячие клавиши должны быть приведены в соответствие с вашими требованиями.

Библиотека содержит две дополнительные клавиатуры: цифровую и символьно–цифровую. Они могут быть использованы так же, как описано в данном примере.

3.9.3 Ввод значений, переключение режимов работы (example 03)

Пример доступен в кадре *pictu_3_chapter_08a.pdl* по нажатию комбинации клавиш *CTRL+W* или щелчку (мыши) на изображенной выше кнопке.
Пример приведен в кадре *pictu_3_chapter_08b.pdl*.

Постановка задачи

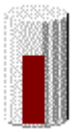
Генерировать в кадре оборудования различные управляющие воздействия без использования мыши. Должны вводиться значения величин и производиться ряд операций переключения.

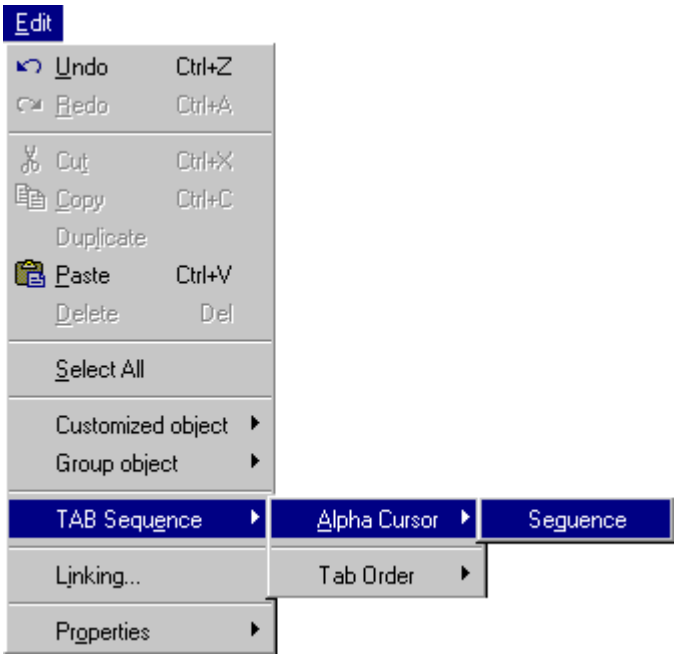
Концепция реализации

Для реализации используются объекты *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*, которым назначены горячие клавиши. В объектах *Smart Objects (Интеллектуальные объекты)* → *I/O Fields (Поля ввода/вывода)* необходимо осуществлять ввод значений, а также открывать и закрывать клапаны.

Реализация в графическом дизайнере

Шаг	Процедура: Реализация в графическом дизайнере
1	В менеджере тегов создайте шесть тегов типа *Signed 16–Bit Value (16–битная величина со знаком)*, которые используются для ввода и последующего хранения данных. В примере это теги от *S16i_pictu_cursor_00* до *S16i_pictu_cursor_05*.
2	Создайте в кадре три *Smart Objects (Интеллектуальные объекты)* → *I/O Fields (Поля ввода/вывода)*, в которых должны вводиться уставки уровня наполнения. В данном примере это объекты *I/O Field1*, *I/O Field2* и *I/O Field3*. Для *I/O Field1* в *диалоге конфигурирования теговое* создайте *соединение с тегом S16i_pictu_cursor_00* и установите *High Limit Value (Величину верхней границы) – 4999* и *Low Limit Value (Величину нижней границы) – 0*. Поступите аналогичным образом с оставшимися *полями ввода/вывода*, но используйте теги *S16i_pictu_cursor_01* или *S16i_pictu_cursor_02*, соответственно. Для *I/O Field3* установите *Upper Limit Value (Величину верхней границы) – 9999*. 

Шаг	Процедура: Реализация в графическом дизайнере
3	Создайте три объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*, которые используются для подтверждения значений, введенных в *полях ввода/вывода*. В данном примере это объекты *ButtonF6*, *ButtonF7* и *ButtonF8*. У *ButtonF6* создайте *процедуру Си* для *Events(События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)*, которая переписывает величину, внесенную в тег *16i_pictu_cursor_00*, в тег *S16i_pictu_cursor_03*. В *Properties (Свойства)* → *Miscellaneous (Разное)* → *Hotkey (Горячая клавиша)* установите для этой *кнопки* горячую клавишу *F6*. Проделайте аналогичные операции с оставшимися *кнопками*. У *ButtonF6* создайте *процедуру Си* для *Properties (Свойства)* → *Geometry (Геометрия)* → *Width (Ширина)*, которая передает фокус этой кнопке. 
4	Создайте три объекта *Standard Objects (Стандартные объекты)* → *Rectangles (Прямоугольники)*, которые представляют введенные величины. В данном примере это объекты *Rectangle9*, *Rectangle10* и *Rectangle11*. Установите *Property (Свойство)* → *Filling (Заполнение)* → *Dynamic Filling (Динамическое заполнение)* – *Yes (Да)*. Для *Properties (Свойства)* → *Filling (Заполнение)* → *Fill Level (Уровень заполнения)* создайте *динамический диалог* для каждого прямоугольника с целью преобразования значения тега в уровень заполнения. Для графического отображения контейнера используются несколько сгруппированных *стандартных объектов*. 
5	Создайте четыре дополнительных объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. Эти кнопки используются для включения и выключения клапанов. В данном примере это объекты *ButtonF9*, *ButtonF10*, *ButtonF11* и *ButtonF12*. Для *Events (События)* → *Mouse(Мышь)* → *Mouse Action(Действие мыши)* создайте *процедуру Си* для каждой кнопки, инвертирующей двоичный тег, представляющий состояние клапана. Каждой *кнопке* назначена горячая клавиша.
6	Создайте в кадре четыре клапана, которые соединены с соответствующими двоичными тегами. Детальное описание процесса создания клапанов можно найти в главе *Displaying and Hiding Information (Отображение и скрытие информации)*, в разделе Displaying and Hiding Objects (Отображение и скрытие объектов) (example 01).

Шаг	Процедура: Реализация в графическом дизайнере
7	Кроме *кнопок*, которым назначены горячие клавиши, все объекты удаляются из порядка обхода клавишей TAB. Командой меню *Edit (Редактирование)* → *TAB Sequence (Последовательность TAB)* → *Alpha Cursor (Альфа–курсор)* → *Sequence (Последовательность)* определяется порядок, в котором с помощью клавиши TAB выбираются *поля ввода/вывода*. 
8	В *Control Center (Центре управления)* определите комбинацию клавиш, которая переключает режим между обходом TAB и альфа–курсором. Щелкните R (правой кнопкой мыши) на пункте *Computer (Компьютер)* и выберите *Properties (Свойства)* из выпадающего меню. В следующем диалоге *Computer List Properties (Свойства списка компьютеров)* щелкните (мышью) по кнопке *Properties (Свойства)*. Выберите вкладку *Graphics Runtime (Графика режима исполнения)*. Для переключения между *TAB Order/Alpha Cursor (Порядок TAB/альфа–курсор)* устанавливается клавиатурная комбинация *SHIFT+A*. Кроме того, задается комбинация *SHIFT+R* для включения и выключения курсора режима исполнения. 

Примечание:

С помощью следующей кнопки или клавиши *ESC*, можно выйти из описанного примера:

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Используемые клавиатурные комбинации и горячие клавиши должны быть приведены в соответствие с вашими требованиями.

Только окрашенные в красный цвет элементы управления обладают функциональностью. Все другие элементы не имеют функций. Вась кадр — это схематичное представление панели оператора *Simatic OP47*.

3.10 Отображение и скрытие информации

Information

Примеры, имеющие отношение к этой теме, доступны в проекте *Project_CreatePicture* по нажатию  (мышью) на *кнопке*, изображенной выше. Примеры приведены в кадре *pictu_3_chapter_09.pdl*.

3.10.1 Отображение и скрытие объектов (example 01)

Во многих кадрах оборудования часто имеет смысл, не отображать постоянно некоторые информационные элементы, а показывать их только при необходимости или при наступлении определенного события.

Постановка задачи

Организовать скрытие пользователем отдельных объектов или групп объектов кадра.

Концепция реализации

Для осуществления этой операции мы используем кадр, в котором отображено несколько клапанов. Каждому клапану назначен объект *Windows Object (Объект Windows)* → *Button (Кнопка)* для управления клапаном, *Standard Object (Стандартный объект)* → *Static Text (Статический текст)* для отображения имени клапана и групповой объект, представляющий состояние клапана. Кроме того, кадр также содержит контейнеры, чьи уровни заполнения отображаются при помощи *Smart Objects (Интеллектуальный объект)* → *I/O Fields (Поля ввода/вывода)*. С помощью трех объектов *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)* все *поля ввода/вывода*, все *кнопки* и все *статические тексты* могут быть отображены или скрыты.

Реализация в проекте WinCC

Шаг	Процедура: Отображение и скрытие объектов
1	В *менеджере тегов* создайте три тега типа *Binary Type (Двоичный тег)*, которые управляют видимостью различных групп объектов. В данном примере используются теги *BINi_pictu_info_12*, *BINi_pictu_info_13*, и *BINi_pictu_info_14*.
2	В *менеджере тегов* создайте дополнительные теги типа *Binary Tag (Двоичный тег)*, которые содержат текущие состояния клапанов. Требуемое число тегов зависит от числа клапанов. В примере используются теги от *BINi_pictu_info_1* до *BINi_pictu_pictu_11* для 11 клапанов.
3	Для отображения открытого клапана создается *Standard Object (Стандартный объект)* → *Polygon (Многоугольник)*, имеющий форму клапана. В поле *Properties (Свойства)* → *Colors (Цвета)* → *Background Color (Цвет фона)* установите цвет *Dark Green (Темно–зеленый)*. 
4	Для отображения закрытого клапана создается объект *Standard Object (Стандартный объект)* → *Polyline (Поли–линия)*, имеющий форму клапана. 

Шаг	Процедура: Отображение и скрытие объектов
5	Создайте два идентичных объекта *Standard Objects (Стандартные объекты)* → *Rectangles (Прямоугольники)* и установите цвет фона кадра в полях *Properties (Свойства)* → *Color (Цвет)* → *Background Color (Цвет фона)*. *Прямоугольники* должны быть чуть больше клапанов, так чтобы закрывать их.
6	Расположите *прямоугольник* и открытый клапан друг над другом и поместите открытый клапан в верхний слой, щелкнув по *кнопке*  . Сгруппируйте оба объекта с помощью команды меню *Edit (Правка)* → *Group Object (Группировать объект)* → *Group (Группировать)*. Для сформированного *группового объекта* создайте *соединение с тегом BINi_pictu_info_1* в поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Display (Отображение)*.
7	Расположите закрытый клапан поверх второго *прямоугольника* и поместите его в верхний слой. Затем расположите созданный на этапе 6 *групповой объект* поверх этого и поместите его в верхний слой. Теперь сгруппируйте эти три объекта. Для построения оставшихся клапанов можно скопировать новый *групповой объект*. Необходимо лишь скорректировать *соединение с тегом*.
8	Для каждого клапана поместите объект *Windows Object (Объект Windows)* → *Button (Кнопка)* и создайте *процедуру Cи* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой клавиши)*, которая инвертирует соответствующий тег.
9	Для каждого клапана создайте *Standard Object (Стандартный объект)* → *Static Text (Статический текст)*, содержащий имя клапана.
10	Сконфигурируйте несколько контейнеров, уровни заполнения которых отображаются при помощи *Smart Objects (Интеллектуальные объекты)* → *I/O Fields (Поля ввода/вывода)*.
11	Разместите три объекта *Windows Object (Объект Windows)* → *Button (Кнопки)*. В данном примере используются объекты *Button12*, *Button13* и *Button14*. У *Button12* создайте *процедуру Cи* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой клавиши)*, которая инвертирует тег *BINi_pictu_info_12*. У оставшихся кнопок создайте *процедуры Cи* для тегов *BINi_pictu_info_13* и *BINi_pictu_info_14* аналогичным образом.
12	Для всех объектов, которые отображаются или скрываются посредством *Button12*, создайте *соединение с тегом BINi_pictu_info_12*. Проделайте то же самое для других объектов. В данном примере *Button12* делает видимым *поля ввода/вывода*, *Button13*, *статические тексты* и *Button14*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Принцип отображения и скрытия объектов может использоваться без изменений.

Способ отображения клапанов можно не менять.

3.10.2 Отображение даты и времени (example 02)

Постановка задачи

Представить различные способы отображения даты и времени.

Концепция реализации

Для реализации будем использовать объекты OCX. Кроме этого используются два объекта *Standard Objects (Стандартные объекты)* → *Static Texts (Статические тексты)*, отображающие дату и время.

Реализация в проекте WinCC

Шаг	Процедура: Отображение даты и времени
1	В меню палитры элементов управления выберите *WinCC Digital/Analog Clock Control (Цифровые/аналоговые часы)*. При этом будет создан индикатор времени, размеры и тип которого остается отрегулировать в соответствии с вашими требованиями.
2	Создайте *Standard Object (Стандартный объект)* → *Static Text (Статический текст)*. В данном примере используется объект *Static Text22*. Для *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* создайте *процедуру Cи*, которая считывает и возвращает текущее время. Установите срабатывание этой операции раз в 1 секунду.
3	Создайте дополнительный *Standard Object (Стандартный объект)* → *Static Text (Статический текст)*. В данном примере используется объект *Static Text23*. Для *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* создайте *процедуру Cи*, которая считывает и возвращает текущую дату.

Процедура Cи для чтения времени

```
#include "apdefap.h"
char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty

{
time_t timer;
struct tm *ptm;
char *p;

time(&timer);
ptm=localtime(&timer);
p=SysMalloc(9);
sprintf(p,"%02d:%02d:%02d",ptm->tm_hour,ptm->tm_min,ptm->tm_sec);
return(p);
}
```

time(timer) возвращает текущее системное время в миллисекундах.

localtime(timer) возвращает указатель на структуру системного времени.

SysMalloc резервирует область памяти.

sprintf создает текст, состоящий из статической части, и нескольких числовых фрагментов.

Процедура Си для чтения даты

```
#include "apdefap.h"
char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty

{
time_t timer;
struct tm *ptm;
char *p;

time(&timer);
ptm=localtime(&timer);
p=SysMalloc(9);
sprintf(p,"%02d:%02d:%02d",ptm->tm_mday,ptm->tm_mon,ptm->tm_year);
return(p);
}
```

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Объект *WinCC Digital/Analog Clock Control* можно перенести в другой проект.

Процедуры Си для объектов *Standard Objects (Стандартные объекты)* → *Static Texts (Статические тексты)* можно использовать в других проектах.

4 Редакторы WinCC (Project_WinCCEditors)

Проект, созданный в данной главе, можно скопировать непосредственно из online–документа на ваш жесткий диск. По умолчанию он будет записан в папку *C:\Configuration_Manual*.


Project_WinCCEditors

В данном проекте приводятся примеры, относящиеся к подсистемам WinCC (и соответствующим им редакторам) *Tag Logging (Архиватор тегов)*, *Alarm Logging (Регистратор аварийных сообщений)* и *Report Designer (Генератор отчетов)*. Примеры приведены в проекте WinCC *Project_WinCCEditors*.

4.1 Регистрация тегов

В режиме исполнения примеры, относящиеся к данной теме, выбираются кнопкой, изображенной выше с использованием (мыши). Эти примеры сконфигурированы в кадрах с *ex_3_chapter_01.pdl* по *ex_3_chapter_01f.pd*.

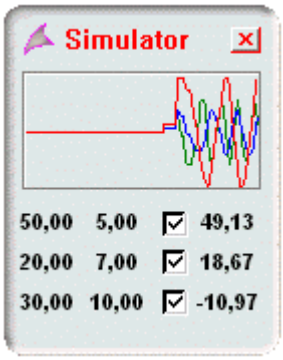
Общая информация

Подсистема регистрации тегов (Tag Logging) содержит функции для использования данных из внешних и внутренних тегов WinCC. Эти данные могут архивироваться различными способами. Отображение данных в режиме исполнения может производиться в виде тренда или таблицы.

Имитация значений процесса

Проект–пример содержит встроенный в проект имитатор для моделирования параметров процесса, архивируемых подсистемой *Tag Logging*. Этот имитатор запускается нажатием на приведенную ниже кнопку.

С помощью данного имитатора можно моделировать синусообразное изменение трех внутренних тегов. Еще один тег представляет собой сумму отдельных значений тегов. Значения тегов отображаются в небольшом окне с трендами.



Ниже окна трендов находятся три строки, каждая из которых содержит поля ввода/вывода для трех тегов. В первом поле устанавливается амплитуда колебания, во втором — частота. Имитация соответствующего тренда останавливается с помощью соответствующего элемента управления. В последнем поле отображается текущее значение тренда. Значения устанавливаются для тегов G64_ex_tlg_01, G64_ex_tlg_02 и G64_ex_tlg_03. Тег G64i_ex_tlg_04 tag представляет собой сумму этих трех тегов. Когда имитатор останавливается, все значения тегов устанавливаются в 0.

4.1.1 Непрерывная циклическая архивация (ex_3_chapter_01.pdl)

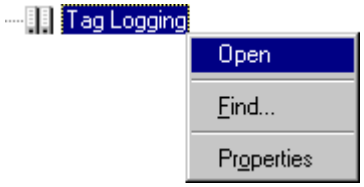
Постановка задачи

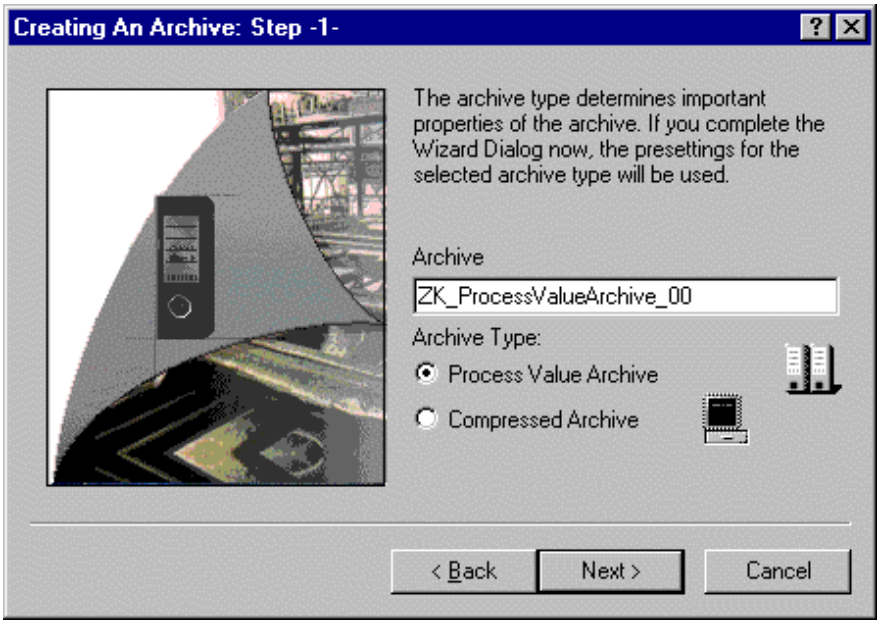
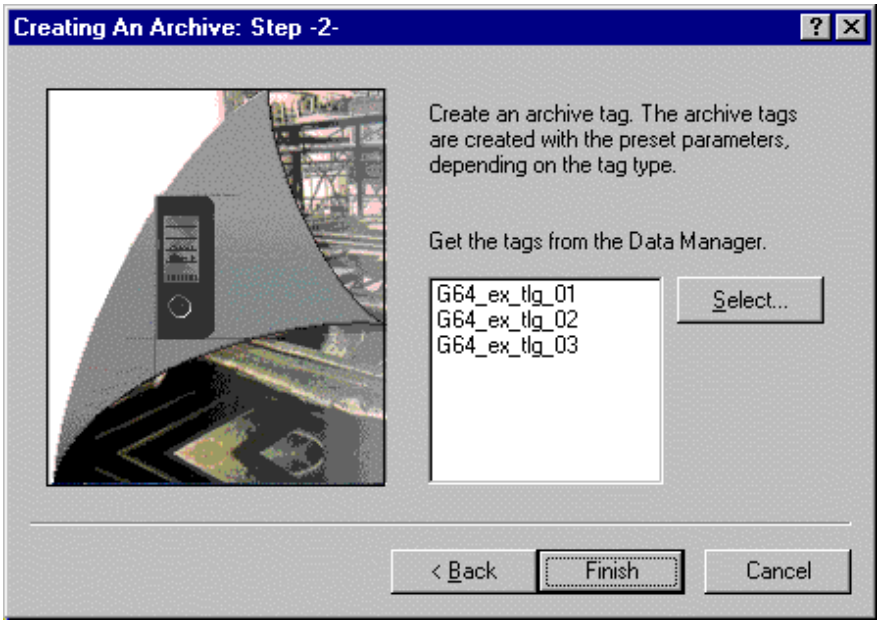
Сохранять значения процесса в архиве при каждом начале цикла. Сохраненные данные графически отображать в режиме исполнения с помощью трендов.

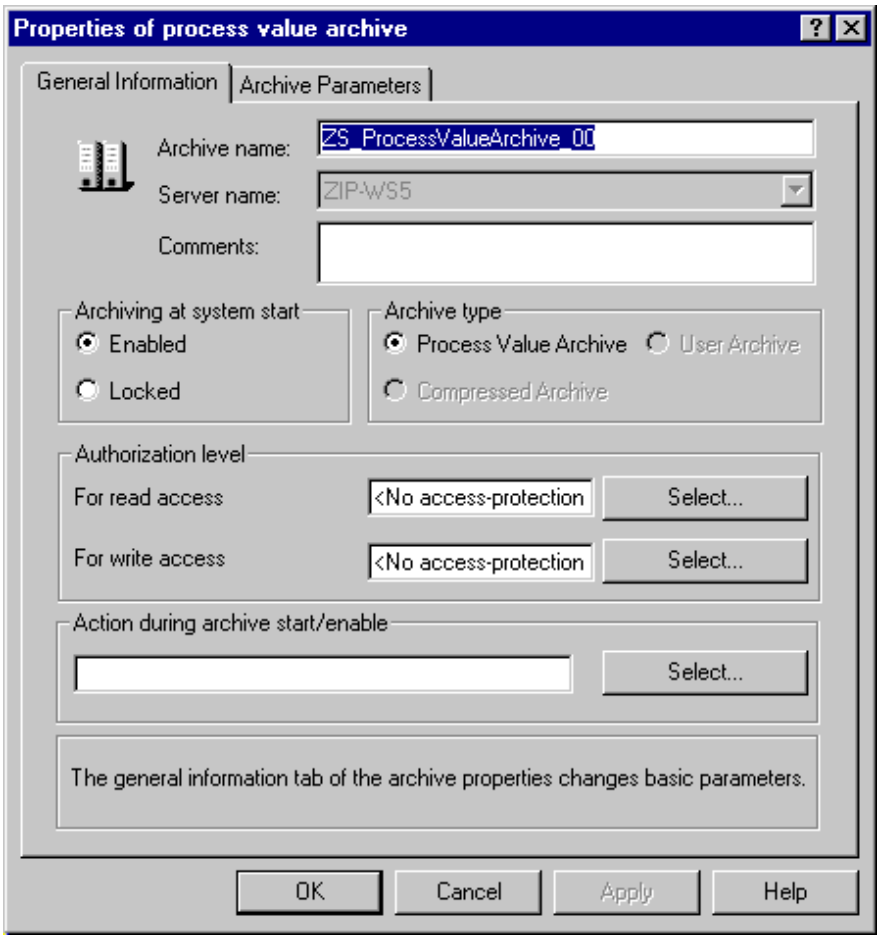
Концепция реализации

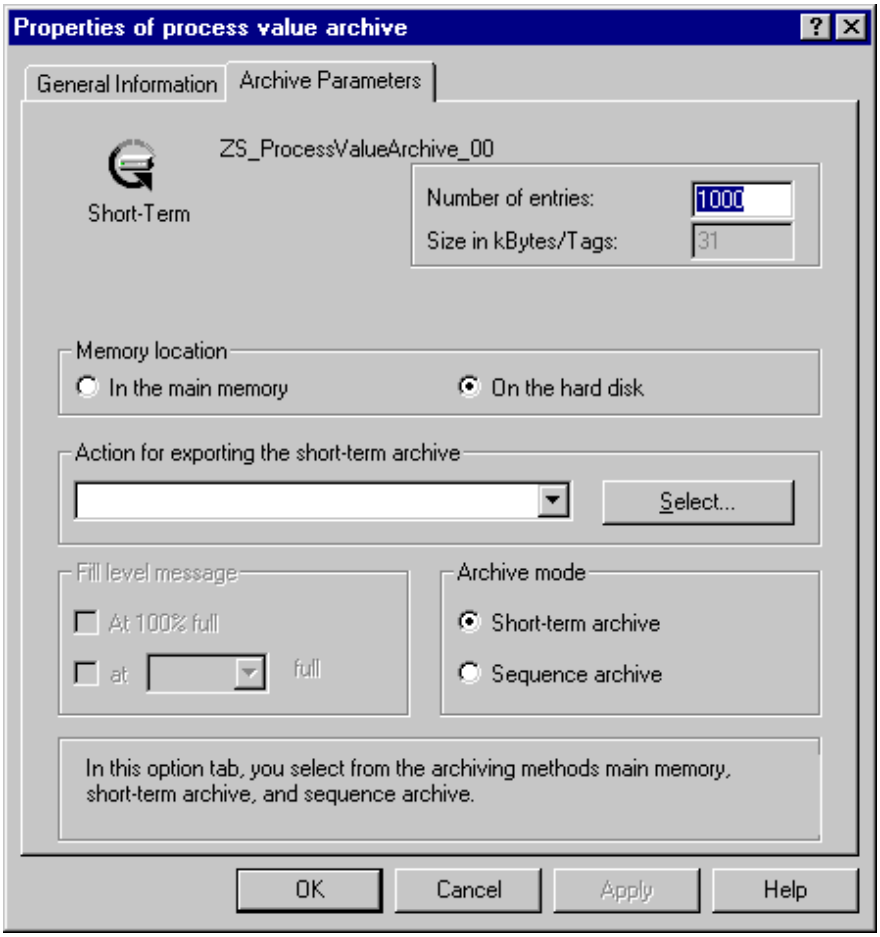
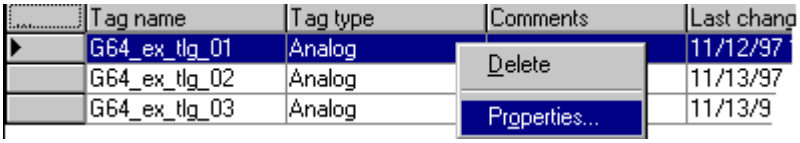
Для архивирования данных, которые должны отображаться, в редакторе *Tag Logging* создается *Cyclic–Continuous Process Value Archive (Непрерывный циклический архив значений процесса)*.
В режиме исполнения архив отображается с помощью специального элемента управления. Этот элемент отображает данные в виде тренда.

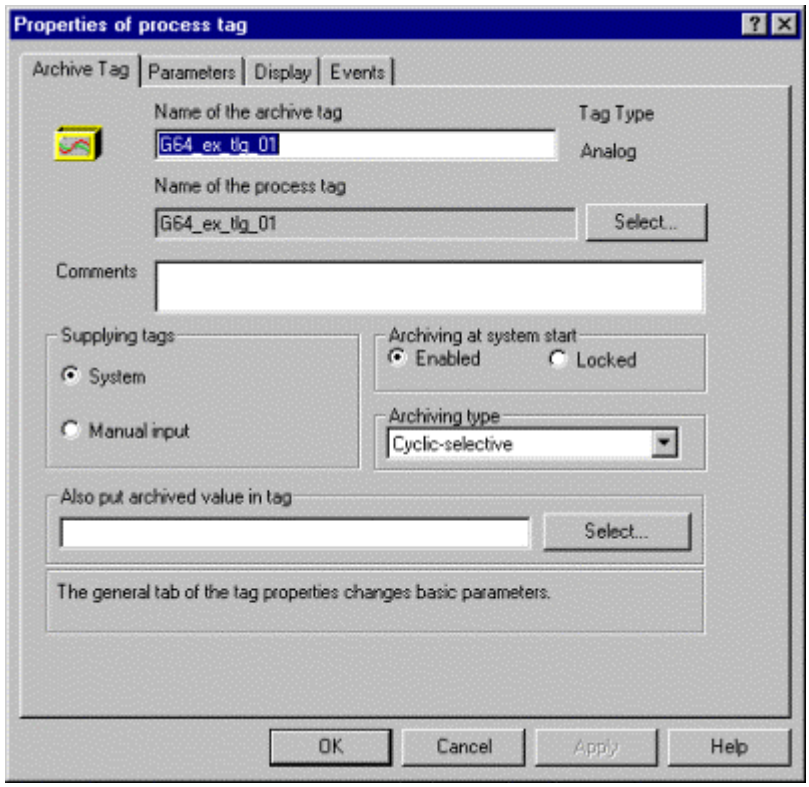
Создание архива значений процесса

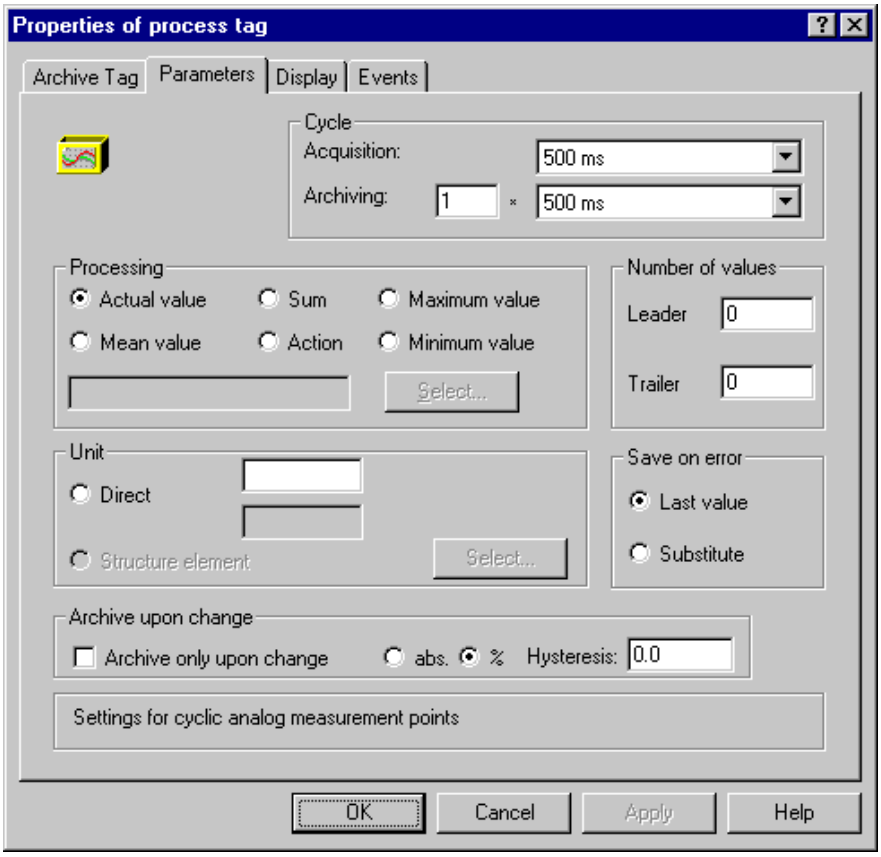
Шаг	Действие: Создание архива значений процесса
1	В *Менеджере тегов* создайте тег для архивирования. В данном примере архивируются теги *G64_ex_tlg_01*, *G64_ex_tlg_02* и *G64_ex_tlg_03*, значения которых изменяются с помощью имитатора.
2	Откройте редактор *Tag Logging*. Это выполняется в окне *проводника WinCC* нажатием R (правой кнопки мыши) на поле *Tag Logging* с последующим выбором *Open (Открыть)* из всплывающего меню. 
3	Создайте новый архив. Нажмите R (правой кнопкой мыши) на поле *Archives (Архивы)* и выберите пункт *Archive Wizard (Мастер архивов)* во всплывающем меню. Этот мастер облегчает пользователю процесс создания нового архива. 

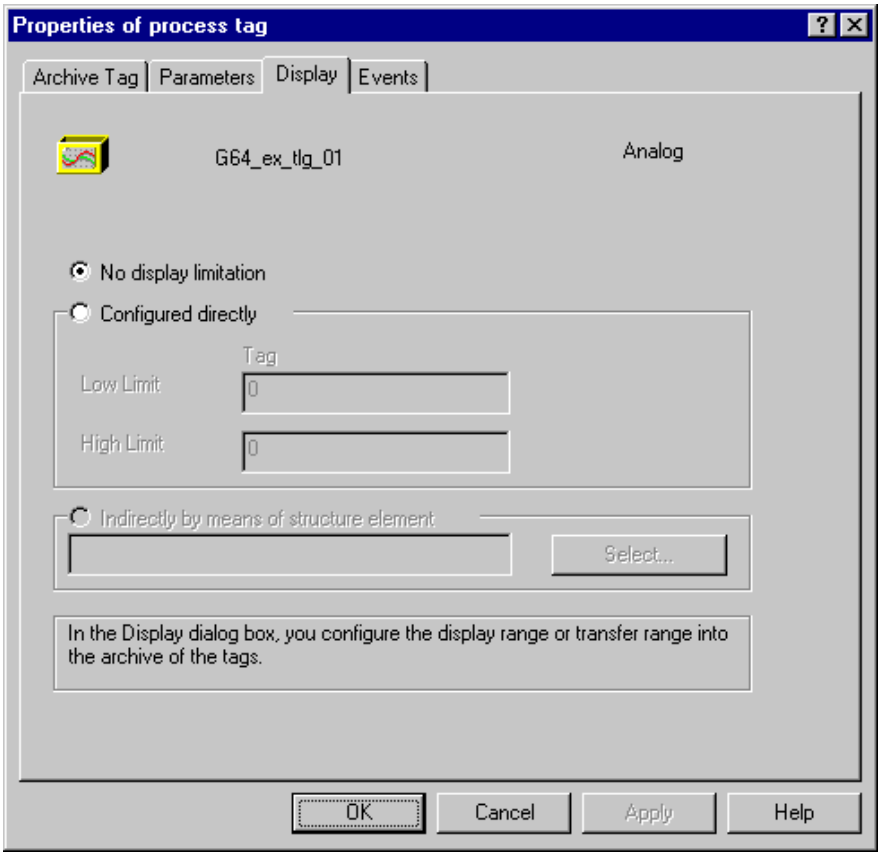
Шаг	Действие: Создание архива значений процесса
4	Начальная страница закрывается нажатием кнопки *Next (Далее)*. На следующей странице в качестве параметра *Archive Type (Тип архива)* укажите вариант *process value archive (архив значений процесса)*. Введите *имя архива (Archive Name)*. В данном примере архив назван *ZK_ProcessValueArchive_00*. Перейдите к следующей странице, нажав на кнопку *Next (Далее)*. 
5	На третьей странице мастера выбираются теги для архивирования. Это производится с помощью кнопки *Select (Выбор)*. В этом примере используются теги *G64_ex_tlg_01*, *G64_ex_tlg_02* и *G64_ex_tlg_03*. Закройте эту страницу мастера, нажав *Finish (Готово)*. 

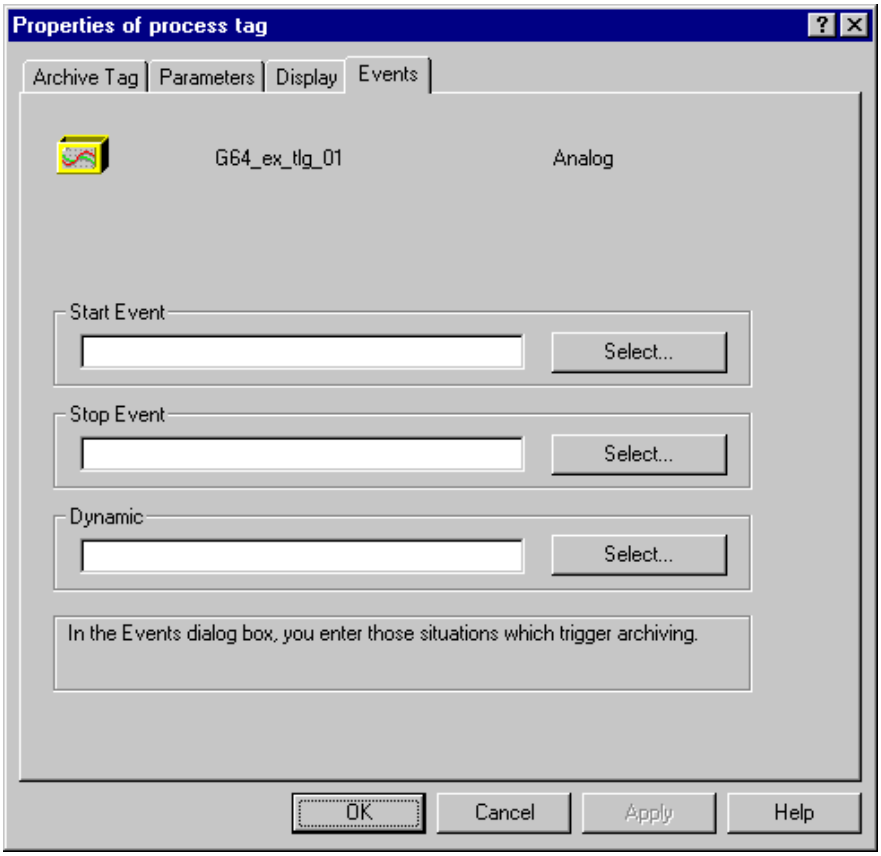
Шаг	Действие: Создание архива значений процесса
6	В правой части окна будет отображено поле только что созданного архива. Нажмите R (правую кнопку мыши) на данном поле и выберите пункт *Properties (Свойства)*; откроется окно свойств данного архива. 
7	На закладке *General Information (Общая информация)* устанавливаются основные параметры архива. Параметры *Archive Name (Имя архива)* и *Archive Type (Тип архива)* уже были выбраны с помощью *мастера архивов*. *Тип архива* изменить позднее нельзя. Параметр *Archiving at System Start (Архивирование с момента запуска)* установлен в *enabled (разрешено)*. Данный параметр инициализирует процесс архивации с момента запуска системы, для этого не требуется отдельной функции. В поле *Authorization Level (Уровень авторизации)* выбрано значение *No Access Protection (Защита доступа отключена)* и на запись, и на чтение. Данные могут использоваться всеми, на них не наложено никакой защиты. При запуске архива не выполняется никаких специальных процедур. Такие процедуры могли бы использоваться, например, для получения информации о статусе архива. 

Шаг	Действие: Создание архива значений процесса
8	На закладке *Archive Parameters (Параметры архива)* устанавливаются дополнительные свойства архива. В качестве размера архива выбрано значение *1000* записей. В качестве *Storage Location (Места хранения)* укажите *On the Hard Disk (На жестком диске)*. Параметр *Archive Type (Тип архива)* установите в *Short–Term Archive (кратковременный архив)*. В качестве *Action for Exporting the Short–Term Archive (Процедура для экспорта кратковременного архива)* может быть выбрана функция, которая будет автоматически выполняться при заполнении кратковременного архива. В данном примере процедура не указана. С данными настройками 1000 записей данных будут архивироваться на жесткий диск. Как только максимальный объем записей будет превышен, самые старые записи начнут заменяться самыми новыми. Закройте диалоговое окно, нажав на *OK*. 
9	Определите свойства отдельных тегов архива. Для этого нажмите R (правой кнопкой мыши) на таблице, приведенной ниже, и выберите пункт *Properties (Свойства)* во всплывающем меню для того, чтобы открыть окно свойств архивного тега. 

Шаг	Действие: Создание архива значений процесса
10	На закладке *Archive Tag (Архивный тег)* устанавливаются основные параметры тега. Соответствующий тег процесса уже был указан в *мастере архивов*. В качестве *Name of the Archive Tag (Имени архивного тега)* укажите нужное имя; в данном примере выбрано имя соответствующего тега процесса. В поле *Supplying Tags (Источник тегов)* выбран вариант *System (Система)*, в поле *Archiving at System Start (Архивирование с момента запуска)* — *Enabled (Разрешено)*. В качестве *Archiving Type (Тип архива)* указано *Cyclic–Continuous (непрерывный циклический)*. При данных настройках опрос данных начинается с момента запуска системы и производится через фиксированные промежутки времени до останова системы. Архивное значение в тег не записывается. 

Шаг	Действие: Создание архива значений процесса
11	На закладке *Parameters (Параметры)* выполняются дополнительные настройки. В поле *Cycle (Цикл)* выберите *Acquisition (Опрос) 500 ms* и установите *Archiving (Архивирование) 1*500 ms*. В поле *Processing (Обработка)* выберите вариант *Actual Value (Действительное значение)*. Значение *Unit (Единицы)* не указывается. В случае ошибки будут сохранены последнее значение. Опция *Archiving upon Change (Архивация по изменению)* не включена. 

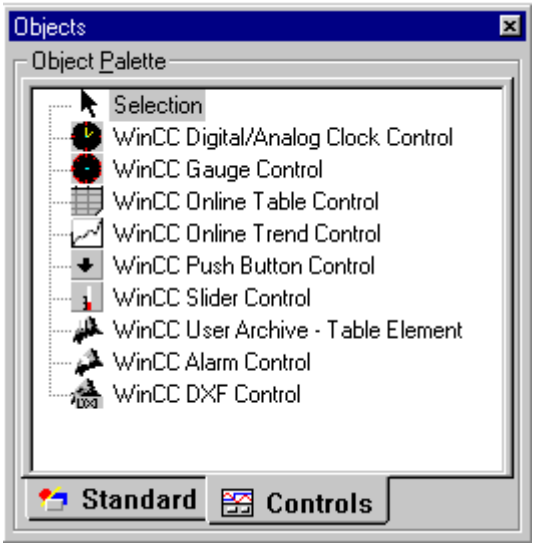
Шаг	Действие: Создание архива значений процесса
12	На закладке *Display (Отображение)* выбирается область допуска для тега в архиве. В данном примере выбран вариант *No Display Limitation (*Отображение б*ез ограничений)*. 

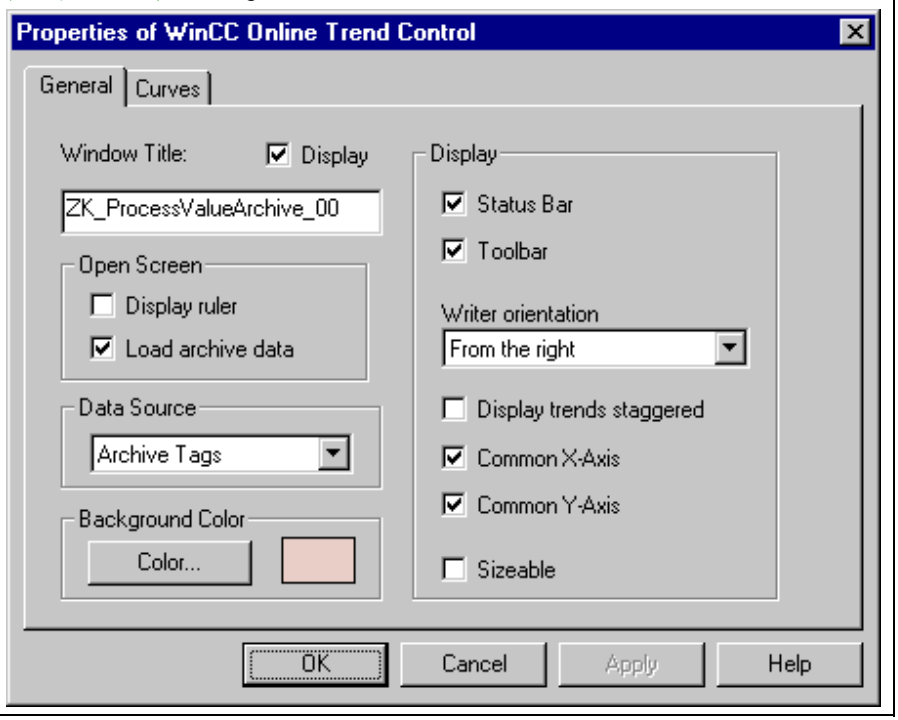
Шаг	Действие: Создание архива значений процесса
13	На закладке *Events (События)* в поле *Dynamic (Динамическое)* в данном примере не назначена процедура для изменения цикла архивации. Закройте диалоговое окно свойств, нажав на *OK*. 
14	Нужно также определить свойства двух других архивных тегов. Для этого повторите шаги с 9–го по 13–й.

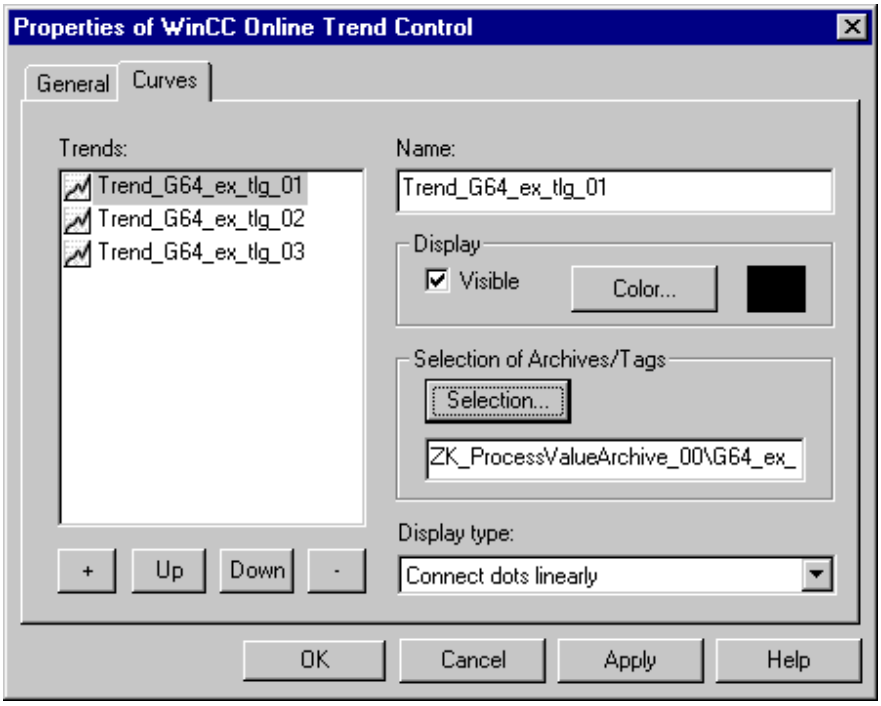
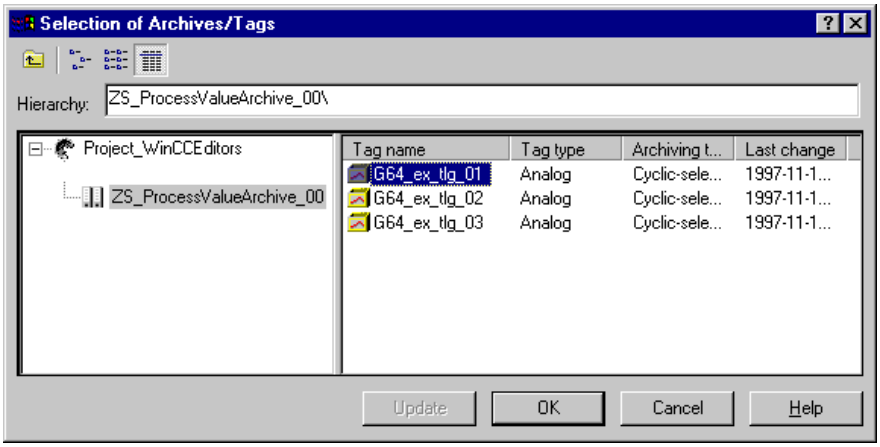
Замечание:

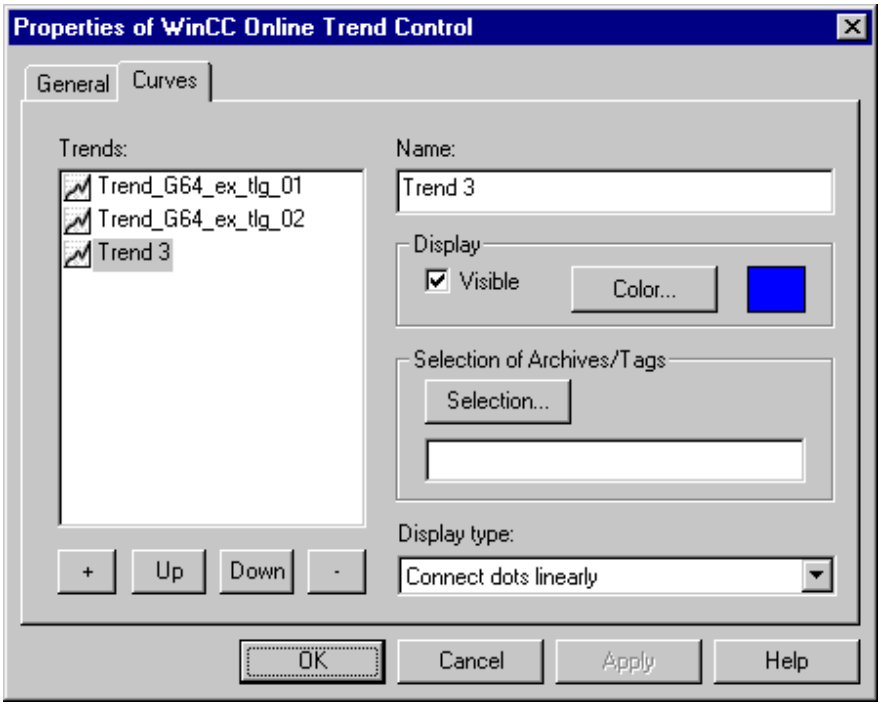
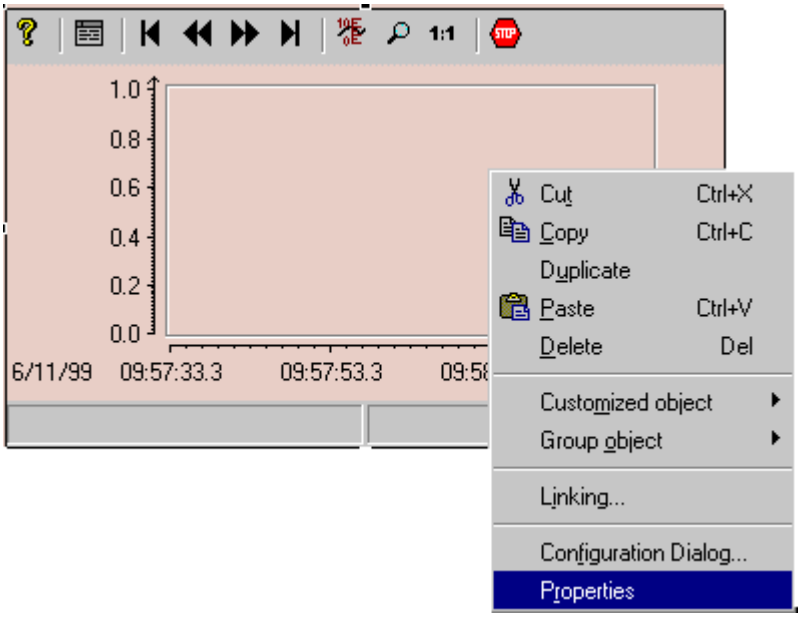
Установки по умолчанию, произведенные мастером архивов при создании архива значений процесса и соответствующие архивные теги могут быть изменены пользователем в пунктах *Archives (Архивы)* –> *Presettings (Предварительные установки)* –> *Process Archive (Архив процесса)* и *Archives (Архивы)* –> *Presettings (Предварительные установки)* –> *Analog Tag (Аналоговый тег)*. Это довольно удобно, если одновременно создается большое число похожих архивов.

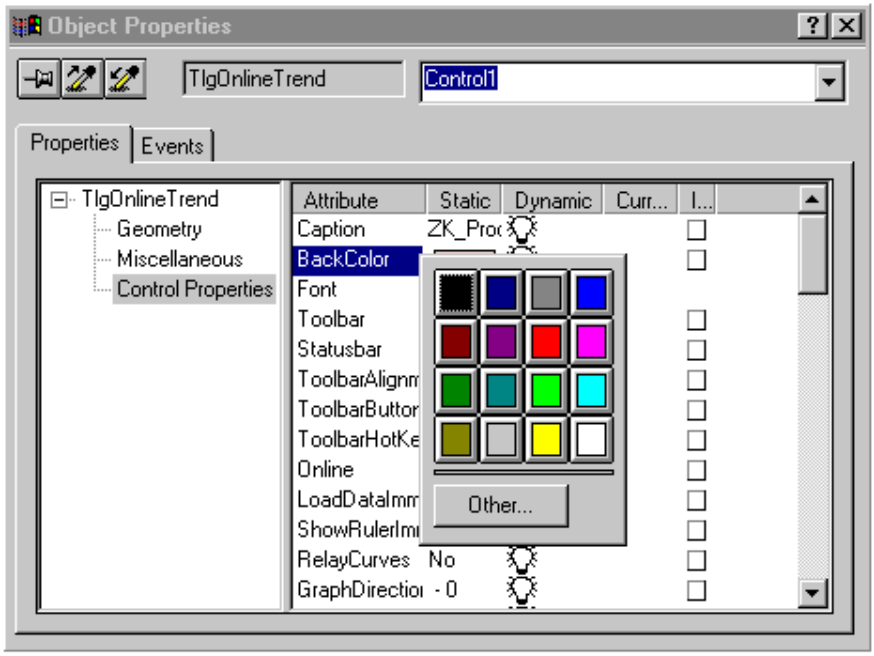
Конфигурирование окна трендов

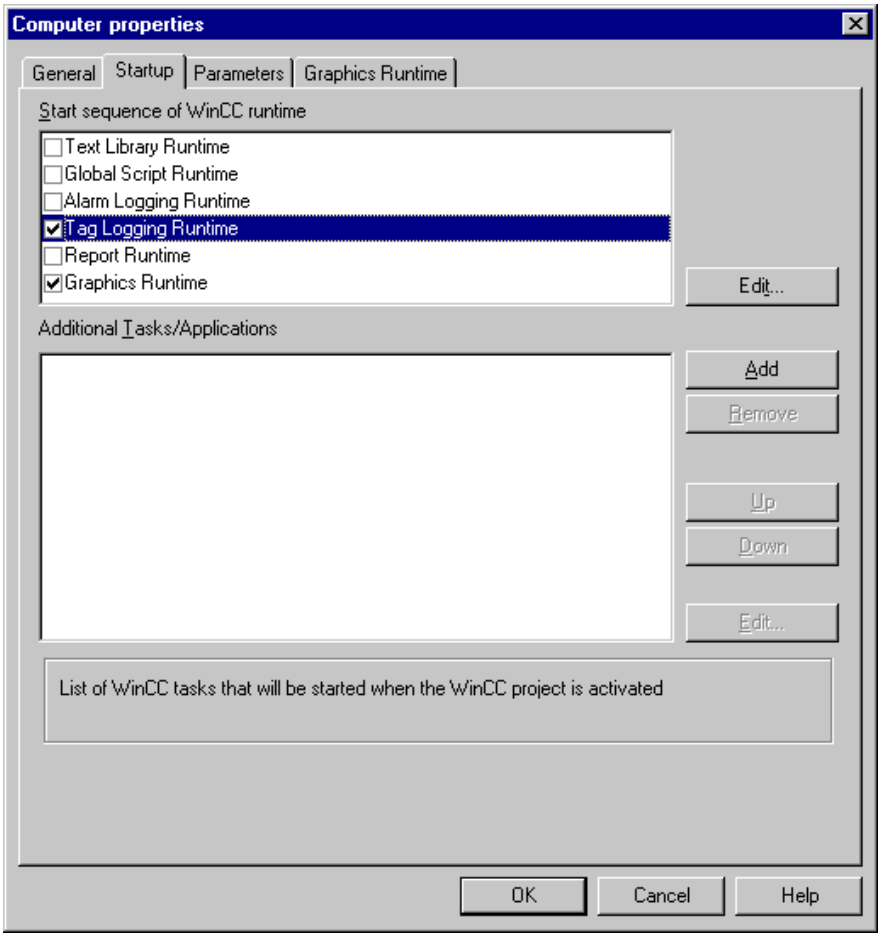
Шаг	Действие: Конфигурирование окна трендов
1	Создайте новый кадр в *графическом дизайнере (Graphics Designer)*. В данном примере это кадр *ex_3_chapter_01.pdl*.
2	Сконфигурируйте элемент управления *WinCC Online Trend Control*, используемый для отображения трендов. Он выбирается в меню *Object Palette (Палитра объектов)* и затем помещается в кадр. 

Шаг	Действие: Конфигурирование окна трендов
3	После помещения экранного элемента в кадр автоматически открывается диалоговое окно конфигурации. На закладке *General Information (Общая информация)* вы можете указать название элемента. В данном примере включена опция *Display (Отображение)*, в качестве *Window Title (Названия окна)* выбрано имя уже созданного архива — *ZK_ProcessValueArchive_00*. В поле *Open Screen (Открытие кадра)* вы можете включить отображение окна линейки (ruler window) при открытии кадра. Окно линейки, таким образом, может быть открыто с помощью соответствующей кнопки на панели инструментов. Включите опцию *Load Data from Archive (Загружать данные из архива)*. Если этого не сделать, элемент будет показывать только те значения, которые были помещены в архив после открытия кадра. В поле *Data Source (*Источник данных*)* вы можете что отображать: значения архивных тегов — *Archive Tags*, или текущих значений — *Online Tags*. При опции *Online Tags* могут отображаться тренды тегов, которые не архивируются. В данном примере выбрана опция *Archive Tags*. С помощью кнопки *Color (Цвет)* выбирается *Background Color (Цвет фона)* окна тренда. Если нужно получить доступ ко всей палитре, это делается с помощью диалогового окна *Object Properties (Свойства объекта)* объекта *Control1*, как показано в Шаге 7. В поле Display в данном примере включены опции *Toolbar (Панель инструментов)* и *Status Bar (Строка статуса)*. В качестве *Writer Orientation (Направления записи)* выбран вариант *From the Right (Справа)*. Также включены опции *Common X Axis (Общие оси X)* и *Common Y Axis (Общие оси Y)*. Размер окна меняться не может. 

Шаг	Действие: Конфигурирование окна трендов
4	На закладке *Curves (Тренды)* отображаются свойства тех трендов, которые должны выводится в окне. Один тренд уже был создан. В данном примере этот тренд переименован в *Trend_G64_ex_tlg_01*. В качестве *Display Type (Типа отображения)* выбран вариант *Connect Dots Linearly (Соединять точки прямыми)*. С помощью кнопки *Selection (Выбор)* архивный тег ассоциируется с трендом. 
5	Откроется окно *Archive/Tag Selection (Выбор архива/тега)*. В левой части выберите нужный архив — в данном примере это *ZK_ProcessValueArchive_00*. В правой части выберите нужный архивный тег (*G64_ex_tlg_01*) из списка тегов в данном архиве. Закройте диалоговое окно, нажав на *OK*. 

Шаг	Действие: Конфигурирование окна трендов
6	Создайте два дополнительных тренда для отображения оставшихся архивных тегов. Новый тренд добавляется в закладку *Trends (Тренды)* нажатием на кнопку +. Изменение их свойств описано в шагах 4 и 5. Используются архивные теги *G64_ex_tlg_02* и *G64_ex_tlg_03*. Окно свойств закрывается кнопкой *OK*. 
7	Выбор цвета фона окна трендов. Для этого нажмите RR (правую кнопку мыши) и затем выберите *Properties (Свойства)* во всплывающем меню для открытия диалогового окна *Object Properties (Свойства объекта)* объекта *Control1*. 

Шаг	Действие: Конфигурирование окна трендов
	В данном примере *BackColor (Цвет фона)* совпадает с цветовой схемой, используемой в проекте. Вы также можете произвести здесь все настройки диалогового окна *WinCC User Archives Table Control Properties (Управляющие свойства таблицы пользовательских архивов WinCC)*. Для некоторых свойств, однако, это не всегда бывает удобно. 

Шаг	Действие: Конфигурирование окна трендов
8	Активация *Tag Logging Runtime (Среды исполнения подсистемы регистрации тегов)*. Нажмите R (правой кнопкой мыши) на поле *Computer (Компьютер)* в *проводнике WinCC* и выберите *Properties (Свойства)* из появившегося меню. Откроется окно *Computer List Properties (Свойства компьютера)*. Нажмите кнопку *Properties (Свойства)* для того чтобы открыть окно свойств локального компьютера. На закладке *Startup (Запуск)* указываются приложения, которые должны быть запущены вместе с системой исполнения. Включите опцию *Tag Logging Runtime (Среда исполнения подсистемы регистрации тегов)*. Диалоговые окна *Computer Properties* и *Computer List Properties* закрываются нажатием на *OK*. 

Замечание относительно диалоговых окон свойств (Properties):

Для задания свойств объекта *WinCC Online Trend Control* можно использовать три диалоговых окна свойств.

Configuration Dialog (Диалог конфигурирования): Это окно открывается автоматически при создании элемента управления. Оно предоставляет пользователю возможность отредактировать основные свойства для быстрого конфигурирования экранного элемента. Оно открывается  D (двойным щелчком мыши) на экранном элементе с нажатой клавишей SHIFT.

Properties Dialog (Диалог свойств): Это окно предоставляет больше возможностей по настройке экранного элемента. Оно открывается  D (двойным щелчком мыши) на элементе кадра.

Object Properties Dialog (Диалог свойств объекта): Это окно является окном по умолчанию для *графического дизайнера*. Оно открывается нажатием  R (правой кнопки) на экранном элементе с последующим выбором *Properties (Свойства)* во всплывающем меню.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

Архивируемые теги должны быть настроены соответствующим образом.

Короткий (быстрый) цикл архивации, выбранный в данном примере, имеет смысл, только если должны отображаться быстро изменяющиеся значения. В большинстве случаев достаточен более медленный цикл обновления. Быстрый цикл архивации приводит к высокой загрузке системы.

4.1.2 Выборочная циклическая архивация (ex_3_chapter_01a.pdl)

Постановка задачи

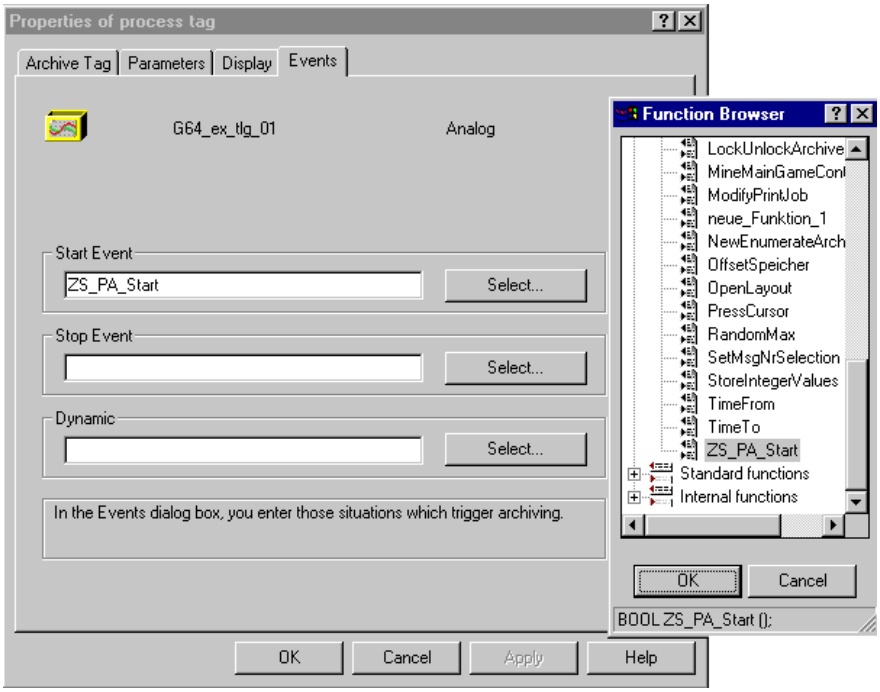
Непрерывно с заданной периодичностью фиксировать в архиве значения процесса. Архивирование должно запускаться и останавливаться по нажатию кнопки. Сохраненные данные должны представляться в режиме исполнения с использованием трендов. Соответствующим образом должны быть сконфигурированы панель инструментов и строка статуса.

Концепция реализации

Для архивирования отображаемых данных в редакторе *Tag Logging* создается *Cyclic–Selective Process Value Archive (Циклический выборочный архив)*.
В режиме исполнения архив отображается с помощью специального элемента управления. Этот объект отображает данные в виде тренда. Требуемая панель инструментов реализуется с помощью объектов *Buttons (Кнопки)*, *Status Displays (Индикаторы состояния)* и *Graphic Objects (Графические объекты)*. Строка статуса выполняется в виде двух кнопок.
Для управления архивом необходимо создать *функцию проекта*, которая будет запускать и останавливать процесс.

Создание архива значений процесса

Шаг	Действие: Создание архива значений процесса
1	В *менеджере тегов* создайте теги, которые должны архивироваться. В данном примере архивируются теги *G64_ex_tlg_01*, *G64_ex_tlg_02* и *G64_ex_tlg_03*, их значения задаются имитатором. Создайте дополнительный двоичный тег (*Binary Tag*), в котором будет храниться текущее состояние архива. В данном примере это тег *BINi_ex_tlg_00*.
2	Создание *функции проекта* для запуска/останова архивации в редакторе *Global Script*. В данном примере это функция *ZS_PA_Start*. Ее описание приведено после данной таблицы.
3	Создание *Process Value Archive (Архива значений процесса)* в редакторе *Tag Logging*. Для этого используйте *мастер архивов*. В данном примере архив назван *ZS_ProcessValueArchive_00*. Для архивации выбраны теги *G64_ex_tlg_01*, *G64_ex_tlg_02* и *G64_ex_tlg_03*.
4	Установка свойств *архива значений процесса*. Размер архива установлен в *1000* записей на закладке *Archive Parameters (Параметры архива)*.

Шаг	Действие: Создание архива значений процесса
5	Изменение свойств *архивных тегов*. Для каждого из трех тегов на закладке *Archive Tag* в качестве параметра *Archiving Type (Тип архивации)* выберите *cyclic–selective (*выборочная циклическая*)*. Этот тип архивации дает возможность назначить на закладке *Events (События)* события *Start Event (Запуск)* и *Stop Event (Останов)*. В данном примере с запуском связана созданная заранее *функция процесса ZS_PA_Start*. Для остальных опций оставлены значения по умолчанию. 

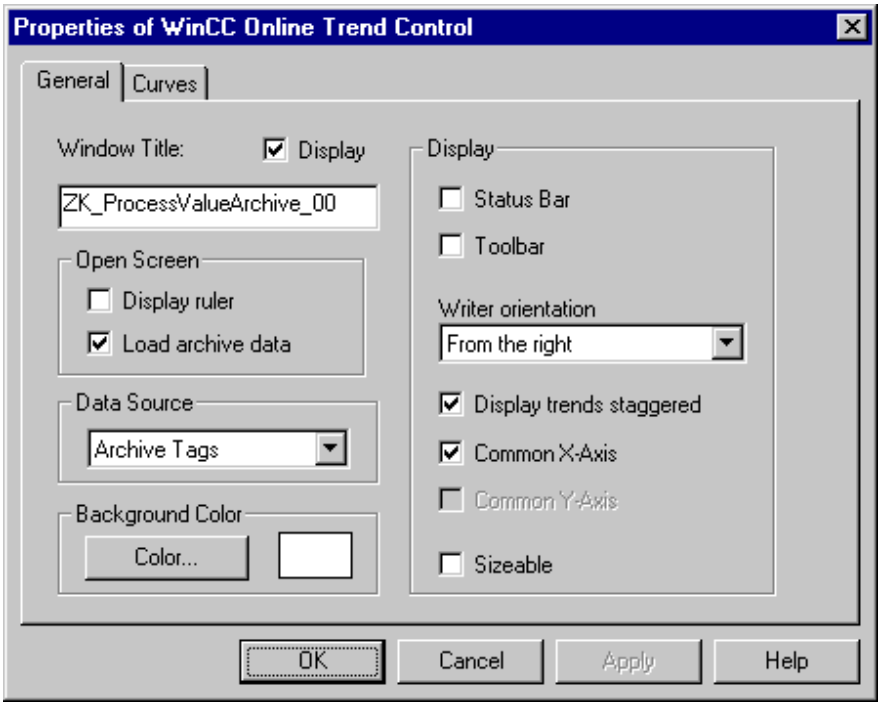
Функция проекта ZS_PA_Start

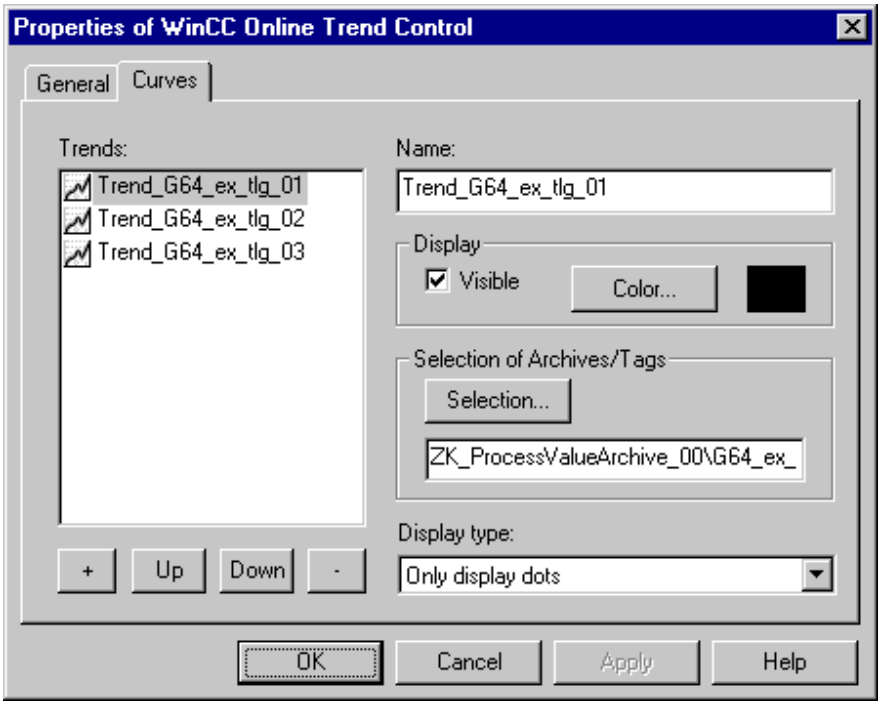
```
BOOL ZS_PA_Start()
{

if (GetTagBit("BINi_ex_tlg_00"))
{
      return TRUE;
}
else
{
      return FALSE;
}

}
```

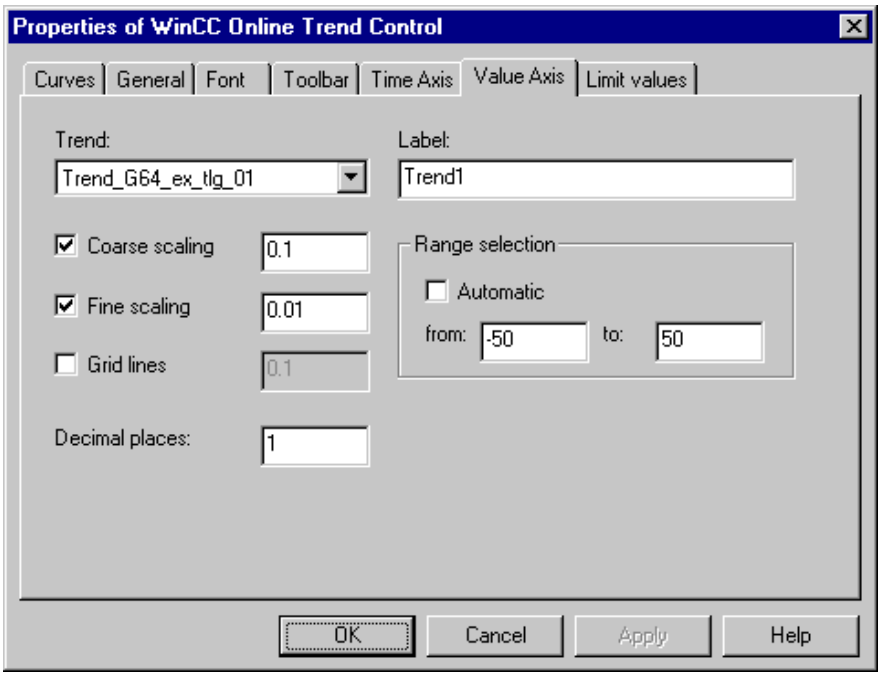
Эта функция возвращает значения *TRUE (Истина)* или *FALSE (Ложь)* в зависимости от статуса двоичного тега *BINi_ex_tlg_00*.

Эта функция вызывается *Tag Logging* в каждом цикле архивации. С помощью возвращаемого значения проверяется, осуществляется архивация или нет. Возвращенное значение *TRUE* запускает архивацию.

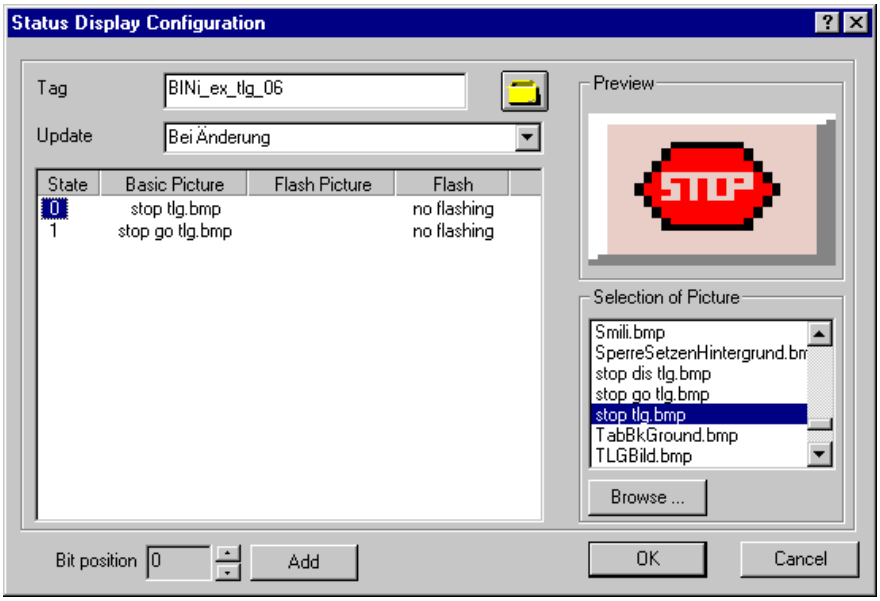
Конфигурирование окна трендов

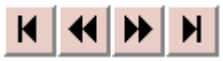
Шаг	Действие: Конфигурирование окна трендов
1	В *графическом дизайнере* создайте новый кадр. В данном примере это кадр *ex_3_chapter_01a.pdl*.
2	Конфигурирование экранного элемента, используемого для отображения трендов — *WinCC Online Trend Control*. Он выбирается из меню *Object Palette (Палитра объектов)* и затем помещается в кадр, после чего автоматически открывается конфигурационное диалоговое окно. На закладке *General Information* вы определяете название элемента. В данном примере опция *Display* не включена, но название окна (*Window Title*) все равно введено. В *процедурах Cu*, созданных позднее, это название окна используется в качестве ссылки на соответствующий экранный элемент. Используется имя уже созданного архива, *ZS_ProcessValueArchive_00*. С помощью кнопки *Color (Цвет)* выбирается цвет фона *(Background Color)* — в данном случае, белый. В данном примере указано, что *Toolbar (Панель инструментов)* и *Status Bar (Строка статуса)* не отображаются. Также выбрана опция *Stagger Trends*, позволяющая отображать каждый тренд отдельно. Для остальных опций оставлены значения по умолчанию. 

Шаг	Действие: Конфигурирование окна трендов
3	На закладке *Trends (Тренды)* производится дополнительная настройка трендов. Создаются три тренда, которым назначены теги с *G64_ex_tlg_01* по *G64_ex_tlg_03* из архива *ZS_ProcessValueArchive_00*. В качестве цвета для всех трех трендов выбран белый, в качестве *Display Type (Тип отображения) — Show only Dots (Точечный)*. Для остальных опций оставлены значения по умолчанию. Окно свойств закрывается нажатием на *OK*. 

Шаг	Действие: Конфигурирование окна трендов
4	Индивидуальная настройка трендов. Для этого используется расширенное диалоговое окно, которое открывается D (двойным щелчком мыши) на экранном элементе. Окно свойств, описанное выше, можно открыть с помощью D (двойного щелчка мыши) и кнопки CTRL. Расширенное окно свойств помимо уже описанных закладок *General Information (Общая информация)* и *Trends (Тренды)* содержит пять дополнительных. В данном примере изменения производятся только на закладке *Value Axis (Ось значений)*. В поле *Trend (Тренд)* выберите *Trend_G64_ex_tlg_01* для изменения свойств именно этого тренда. В поле *Label (Метка)* введено *Trend1*. Параметр *Range Selection (Выбор диапазона)* не установлен в автоматический режим, границы заданы с *–50* по *50*. Для остальных опций оставлены значения по умолчанию. Точно таким же образом редактируются свойства оставшихся трендов. Окно закрывается нажатием на *OK*. 

Конфигурирование панели инструментов и строки статуса

Шаг	Действие: Конфигурирование панели инструментов и строки статуса
1	В *менеджере тегов* создайте внутренний двоичный тег. В данном примере это тег *BINi_ex_tlg_06*.
2	Для управления обновлением нужно сконфигурировать *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*. В данном примере используется объект *Status Display5*. С помощью *конфигурационного диалога* объект соединяется с тегом *BINi_ex_tlg_06* и изменяется при изменении данного параметра. Создаются состояния *0* и *1*, в данном примере этим состояниям назначены пиктограммы *stop tlg.bmp* и *stop go tlg.bmp*. Окно закрывается нажатием на *OK*. 
3	У только что сконфигурированного объекта *Status Display5* создайте *процедуру Си* для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Эта *процедура Си* имитирует нажатие кнопки Stop/Go (Останов/Запуск) на стандартной панели инструмента элемента. Состояние тега *BINi_ex_tlg_06* инвертируется для индикации изменения состояния процесса. Нулевое значение тега соответствует активизированному обновлению. Значение тега *BINi_ex_tlg_06* всегда равно нулю при открытии кадра, т.к. обновление окна трендов в этом случае всегда запущено. Это реализовано в *прямом соединении* для события *Events (События)* → *Miscellaneous (Разное)* → *Open Picture (Открытие кадра)* объекта кадра *ex_3_chapter_01a.pdl*.

Шаг	Действие: Конфигурирование панели инструментов и строки статуса
4	В соответствии с описанием, приведенном в шаге 2, сконфигурируйте *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*. В данном примере это объект *Status Display6*. Этот объект используется для управления архивацией. Данный объект соединен с тегом *BINi_ex_tlg_00*, созданным в предыдущем разделе. Соответственно, используются другие пиктограммы (*Archive.bmp* / *Archive inv.bmp*). Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создается новая а *процедура Cи*. Эта процедура инвертирует тег *BINi_ex_tlg_00*. Данный тег используется для отображения измененного статуса архивации и для передачи этой информации архиву с помощью *функции проекта ZS_PA_Start*. 
5	Для навигации по архиву при остановленном процессе обновления используются четыре кнопки навигации стандартной панели инструментов. В кадр помещены четыре *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*; в данном примере это объекты *Button4*, *Button7*, *Button8* и *Button11*. У каждого из этих объектов для события *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* создается *процедура Cи*. Эти процедуры имитируют нажатия кнопок на стандартной панели инструментов. Дополнительно требуется *Smart Object (Интеллектуальный объект)* → *Graphic Object (Графический объект)*, который помещается поверх этих кнопок и делает их недоступными при включенном обновлении. В данном примере это объект *Graphic Object2*. Он отображает четыре отключенные кнопки (*Pfeile dis.bmp*). В пункте *Properties (Свойства)* → *Miscellaneous (Разное)* → *Display (Отображение)* создайте *Dynamic Dialog (Динамический диалог)*. Это окно контролирует видимость объекта в зависимости от значения тега *BINi_ex_tlg_06*, который содержит информацию об обновлении экранного элемента. 
6	Для отображения строки статуса используются два объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*; в данном примере это кнопки *Button5* и *Button6*. Кнопки используются в качестве текстовых полей, так как им легко назначить 3D рамку, и, следовательно, не требуется использовать дополнительных элементов. Для *Button5* в пункте *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* создайте *процедуру Cи*. Эта процедура возвращает либо текст *Archiving Started (Архивирование запущено)*, либо *Archiving Stopped (Архивирование остановлено)*, в зависимости от значения тега *BINi_ex_tlg_00*. *Процедура Cи* используется вместо *динамического диалога* для поддержки переключения языков. Соответствующим образом настройте *Button6* для тега *BINi_ex_tlg_06*. 

Замечание:

Реализация кнопок выбора времени и предварительного просмотра (print preview) подробно описана в разделе Печать окна трендов в режиме исполнения главы *Report Designer (Дизайнер отчетов)* (ex_3_chapter_01a.pdl).

Процедура Си для кнопки запуска/останова (Status Display5)

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszI
{

TlgTrendWindowPressStartStopButton("ZS_ProcessValueArchive_00");

SetTagBit("BINi_ex_tlg_06",(SHORT)!GetTagBit("BINi_ex_tlg_06"));

}
```

Вызов стандартной функции *TlgTrendWindowPressStartStopButton* эквивалентен нажатию кнопки запуска/останова на стандартной панели инструментов. Для идентификации элемента управления функции передается текст. Этот текст представляет собой имя окна, которое было указано при конфигурировании объекта. В данном примере это текст *ZS_ProcessValueArchive_00*.

Инвертирование тега *BINi_ex_tlg_06* для запоминания текущего состояния обновления.

Процедура Си для навигационной кнопки запуска (Button4)

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

TlgTrendWindowPressFirstButton("ZS_ProcessValueArchive_00");

}
```

Вызов этой стандартной функции равнозначен нажатию кнопки *First Data Record (Первая запись)* на стандартной панели инструментов. Остальные кнопки используют следующие функции:

- *TlgTrendWindowPressPrevButton*
- *TlgTrendWindowPressNextButton*
- *TlgTrendWindowPressLastButton*

Замечание:

Для каждой кнопки стандартной панели инструментов *WinCC Online Trend Control* существует соответствующая стандартная функция, имитирующая нажатие кнопки.

Процедура Си для отображения текста строки статуса (Button5)

```
#include "apdefap.h"
 char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

char start[40] = "";
char stop [40] = "";

switch(GetLanguage())
{
case 1031 : strcpy(start,"   Archivierung gestartet... ");
            strcpy(stop, "   Archivierung gestoppt... ");
            break;
case 1033 : strcpy(start,"   Archiving started... ");
            strcpy(stop, "   Archiving stoped... ");
            break;
case 1036 : strcpy(start,"   Archivage démarré...");
            strcpy(stop, "   Archivage arrêté...");
            break;
default   : strcpy(start,"   Archivierung gestartet... ");
            strcpy(stop, "   Archivierung gestoppt... ");
            break;
}

if (GetTagBit("BINi_ex_tlg_00"))
{
  return start;
}
else
{
  return stop;
}

}
```

Создаются два текстовых тега. В данные теги, в зависимости от выбранного языка, копируется текст, соответствующий запуску или останову архивации. Выбранный в данный момент язык определяется функцией *GetLanguage()*.

В зависимости от значения тега *BINi_ex_tlg_00* возвращается тег *start* или *stop*. Процедура выполняется при изменении тега *BINi_ex_tlg_00*.

Замечание:

В следующем примере реализована строка статуса с отдельными объектами. В познавательных целях для управления строкой статуса вместо *процедуры Си* используется *динамический диалог*.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

Архивируемые теги должны быть настроены соответствующим образом.

Короткий (быстрый) цикл архивации, выбранный в данном примере, имеет смысл, только если должны отображаться быстро изменяющиеся значения. В большинстве случаев достаточен более медленный цикл обновления. Быстрый цикл архивации приводит к высокой загрузке системы.

Запуск или останов архивации может производиться в зависимости от различных событий, это не обязательно должно быть нажатие кнопки.

Внешний вид элементов может быть настроен в соответствии с вашими требованиями. Это относится и к строке статуса.

Тип отображения был выбран для наилучшего отображения интервалов, в течение которых архивация не производилась. Во всех других типах отображения точки соединяются линиями. Это означает, что интервалы, в течение которых архивация не производилась, отмечаются прямыми.

4.1.3 Архивация при превышении значения (ex_3_chapter_01b.pdl)

Постановка задачи

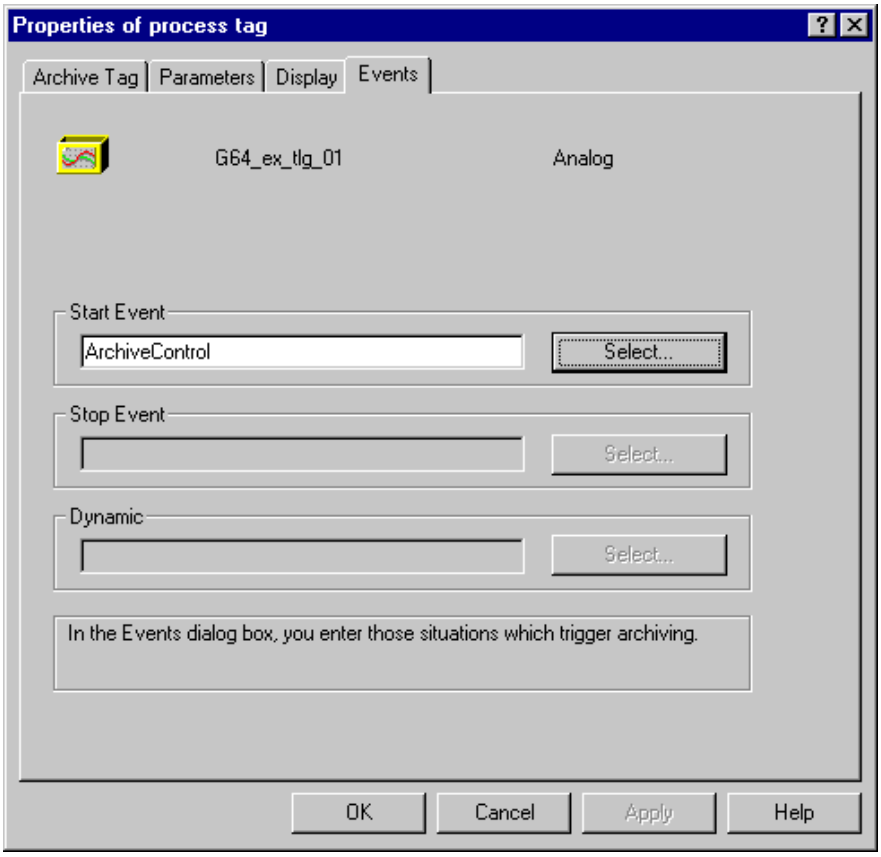
Фиксировать в архиве значения процесса при превышении определенной уставки. Сохраненные значения должны отображаться в виде таблицы. Хронологический процесс изменения этого значения процесса должен иллюстрироваться трендом. Должны быть сконфигурированы панель инструментов и строка статуса.

Концепция реализации

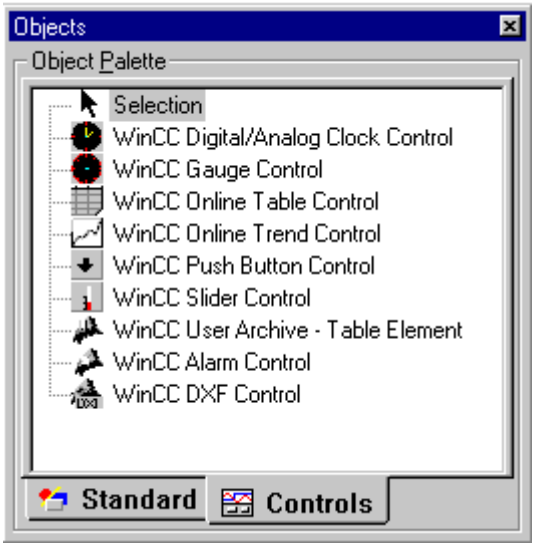
Для архивирования данных в редакторе *Tag Logging* создается *Acyclic Process Value Archive (Ациклический архив значений процесса)*.
В режиме исполнения архив отображается с помощью специального экранного элемента, представляющего данные в виде таблицы. Тренд значения процесса отображается с помощью другого экранного элемента. Панель инструментов реализована с помощью объектов *Button (Кнопка)*, *Status Display (Индикатор состояния)* и *Graphic Object (Графический объект)*. Строка статуса реализована с помощью *кнопки*.
Для управления архивом создается *функция проекта*. Эта функция включает архивирование при выходе значения процесса за уставку.

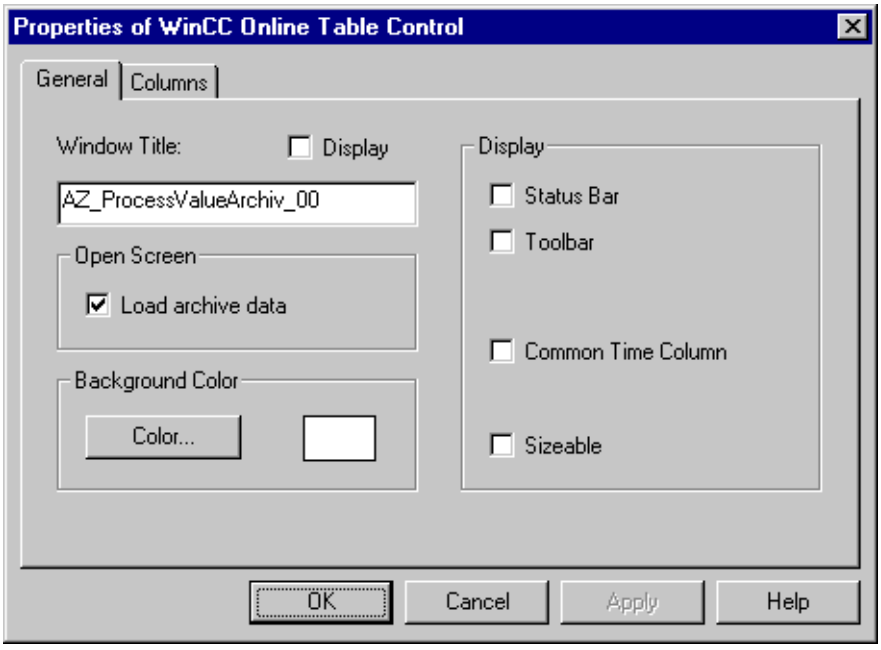
Создание архива значений процесса

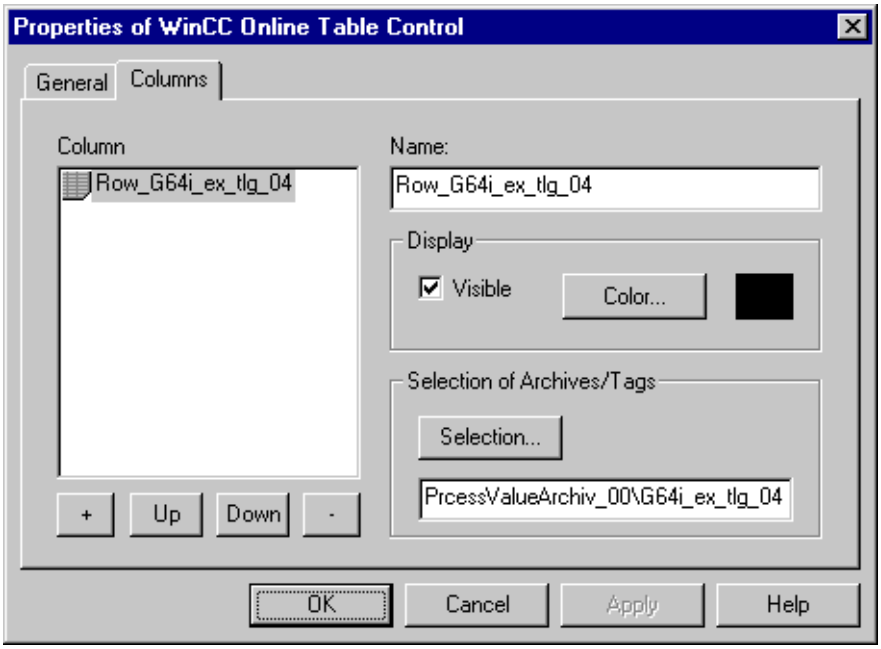
Шаг	Действие: Создание архива значений процесса
1	В *менеджере тегов* создайте два тега. Один тег содержит сумму значений, формируемых имитатором. В данном примере это тег *G64I_ext_lgI_04*. Другой тег архивируется при превышении уставки, это тег *G64i_ex_tlg_08*.
3	Создание *архива значений процесса* в редакторе *Tag Logging*. Воспользуйтесь *мастером архивов*. В данном примере архив назван *AZ_ProcessValueArchive_00*. Для архивирования выбран тег *G64i_ex_tlg_08*.
4	Настройка свойств *архива значений процесса*. Длина архива устанавливается в *25* записей на закладке *Archive Parameters (Параметры архива)*. Для остальных опций оставлены значения по умолчанию.

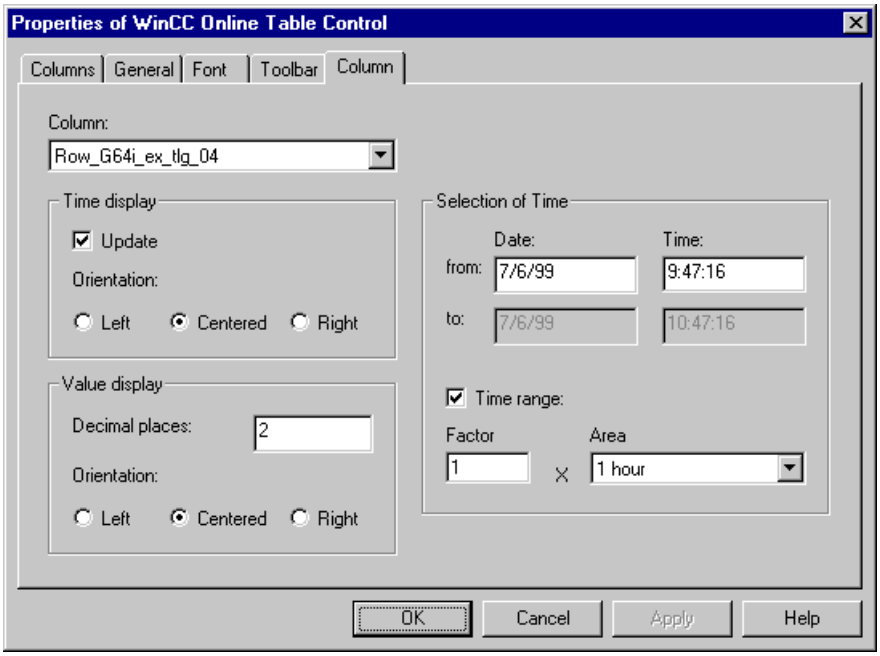
Шаг	Действие: Создание архива значений процесса
5	Настройка свойств *архивных тегов*. На закладке *Archive Tag (Архивные теги)* в качестве *Archiving Type (Тип архивирования)* выберите вариант *acyclic (ациклический)*. Данный тип архива служит для архивирования значений при изменении архивного тега. Для остальных опций оставлены значения по умолчанию. 

Конфигурирование окна таблицы

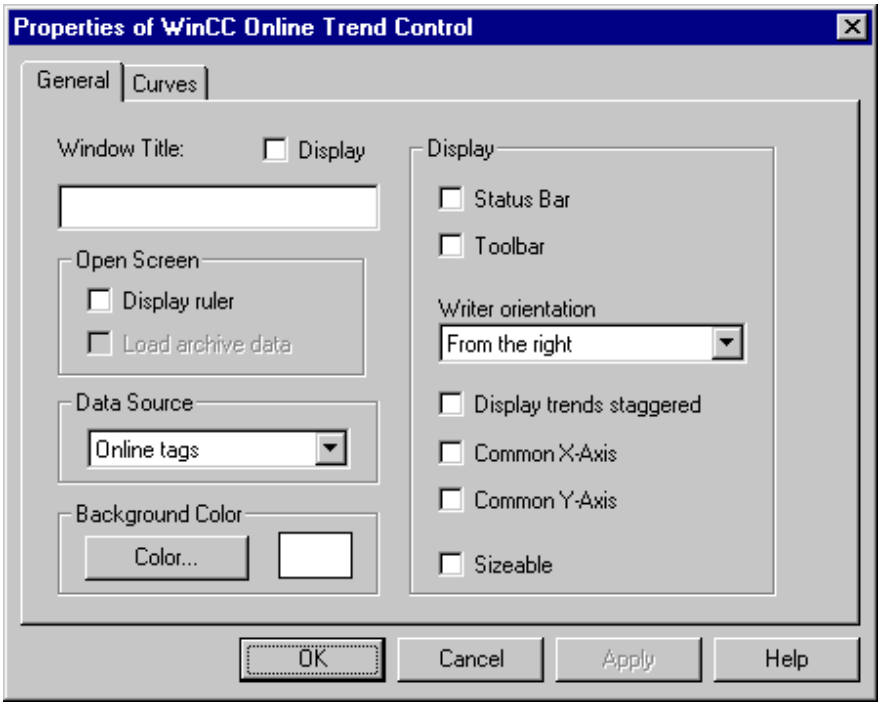
Шаг	Действие: Конфигурирование окна таблицы
1	Создайте новый кадр в *графическом дизайнере*. В данном примере это кадр *ex_3_chapter_01b.pd*.
2	Конфигурирование экранного элемента, используемого для отображения таблицы. Это элемент *WinCC Online Table Control*. Выберите его в *Object Palette (Палитра объектов)* и поместите в кадр. 

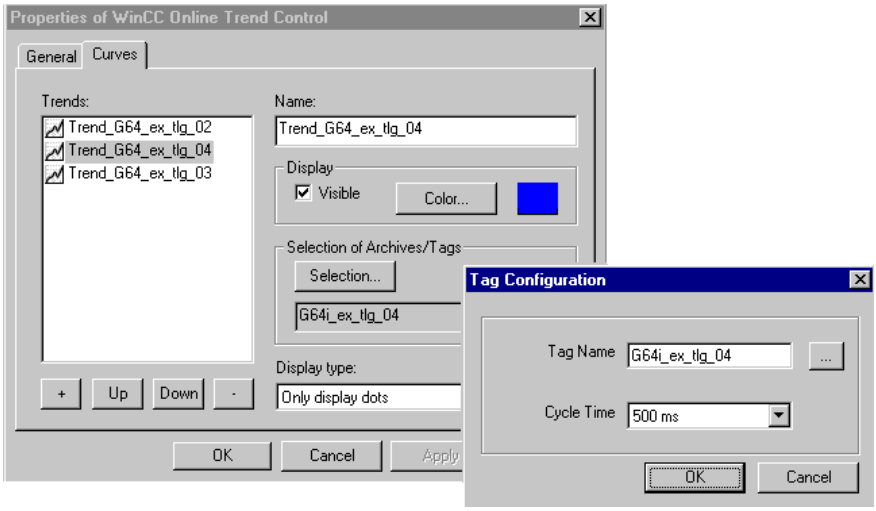
Шаг	Действие: Конфигурирование окна таблицы
3	После помещения экранного элемента в кадр автоматически открывается окно свойств *WinCC Online Table Control Properties*. На закладке *General Information (Общая информация)* вы можете указать название. В данном примере опция *Display (Отображение)* отключена, но *Window Title (Заголовок окна)* все же указан. В созданной позднее *процедуре Cи* это имя используется для идентификации элемента управления. В данном случае выбрано имя уже созданного архива *AZ_ProcessValueArchive_00*. С помощью кнопки *Color (Цвет)* в качестве *Background Color (Цвета фона)* окна таблицы выбран белый цвет. В области *Display (Отображение)* все опции отключены, поэтому панель инструментов и строка статуса отображаться не будут. 

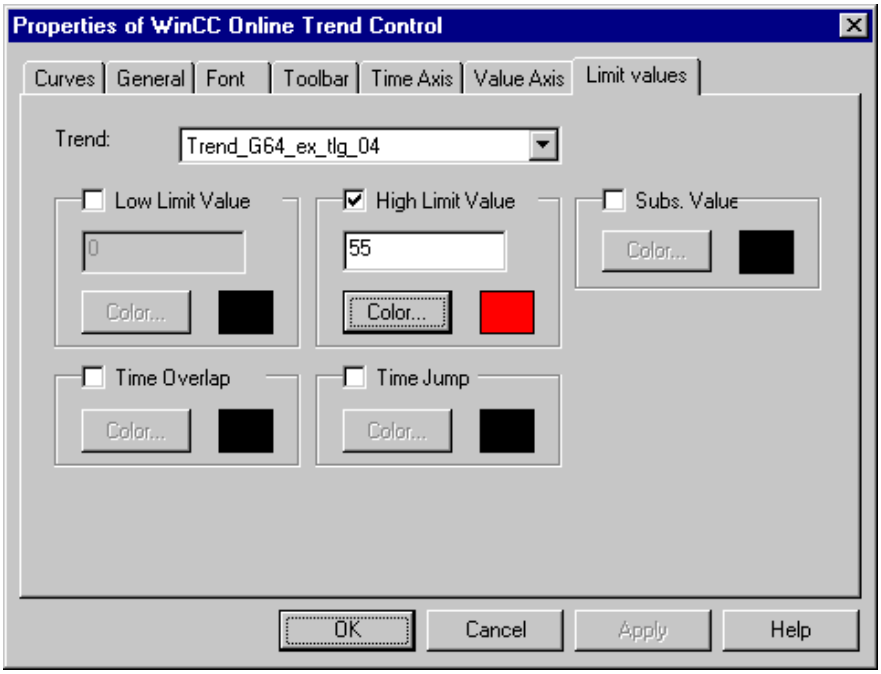
Шаг	Действие: Конфигурирование окна таблицы
4	На закладке *Columns (Столбцы)* указываются столбцы таблицы. В данном примере достаточно только одного столбца. Один столбец уже создан; он переименован в *Row_G64i_ex_tlg_08*. С помощью кнопки *Selection (Выбор)* столбцу назначается *архивный тег*. В данном примере это тег *G64i_ex_tlg_08* уже созданного архива *AZ_ProcessValueArchive_00*. Окно закрывается нажатием на *OK*. 

Шаг	Действие: Конфигурирование окна таблицы
5	Индивидуальная настройка столбцов. Для этого используется расширенное диалоговое окно, которое открывается D (двойным щелчком мыши) на экранном элементе. Окно свойств, описанное выше, можно открыть с помощью D (двойного щелчка мыши) и клавиши CTRL. Расширенное окно свойств в дополнение к уже описанным закладкам *General Information (Общая информация)* и *Columns (Столбцы)* содержит еще три. В данном примере изменения производятся только на закладке *Column (Столбец)*. Атрибут *Format (Формат)* объекта *Time Display (Индикатор времени)* установлен в *hh:mm:ss*. В поле *Orientation (Ориентация)* выбран вариант *Centered (По центру)*. В поле *Time Selection (Выбор времени)* опция *Time Range (Период времени)* включена, но *Range (Период)* установлен в *1 X 1 Час*. Для остальных опций оставлены значения по умолчанию. Окно закрывается нажатием на *OK*. 
6	Запись тега при выходе за уставку. Это осуществляется процедурой Си: нажатие R (правой кнопки мыши) → *Properties (Свойства)* → *Geometry (Геометрия)* → *Position X (Координата X)*. Процедура Си создается, потому что требуется изменение свойства, а само свойство не является динамическим.

Конфигурирование окна трендов

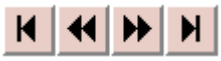
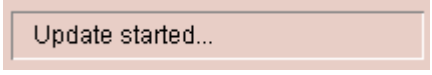
Шаг	Действие: Конфигурирование окна трендов
1	Конфигурирование элемента управления, используемого для отображения тренда. Это объект *WinCC Online Trend Control*. Выберите его в *Object Palette (Палитра объектов)* и поместите в кадр. Окно настроек откроется автоматически. На закладке *General Information (Общая информация)* укажите, что объект должен отображаться без заголовка окна. Для этого отключите опцию *Display*. В поле *Data Supply (Источник данных)* выбран вариант *Online Tag (Текущее значение тега)*, поэтому для отображения хронологического процесса не нужно создавать отдельного архива. Просто выберите требуемый внешний или внутренний тег. Буферизация значений тегов, необходимая для отображения, выполняется самим объектом. С помощью кнопки *Color (Цвет)* в качестве *Background Color (Цвет фона)* окна таблицы выбран белый. В области *Display* все опции отключены. 

Шаг	Действие: Конфигурирование окна трендов
2	На закладке *Trends (Тренды)* производятся индивидуальные настройки трендов. Тренд переименован в *Trend_G64i_ex_tlg_04*. В качестве *Color (цвета)* выбран синий, атрибут *Display Type (Тип отображения)* установлен в *Interpolate Trend (Интерполирование)*. По кнопке *Selection (Выбор)* открывается окно *Tag Configuration (Конфигурация тегов)*. В данном окне настраиваются отображаемые теги. В данном случае, это не *архивный тег*, а так называемый *онлайновый* (внутренний или внешний). Также здесь устанавливается время цикла *Cycle Time*. В данном примере цикл тега *G64i_ex_tlg_04* равен *500 ms*. Окно закрывается нажатием на *OK*. 

Шаг	Действие: Конфигурирование окна трендов
3	Цветовая индикация выхода за уставку. Для этого используется расширенное окно свойств, которое открывается D (двойным щелчком мыши) на экранном элементе. Расширенное окно свойств в добавление к уже описанным закладкам *General Information (Общая информация)* и *Trends (Тренды)* содержат пять дополнительных. На закладке *Value Axis (Ось значений)* в поле *Range Selection (Выбор интервала)* выбрано значение *Automatic (Автоматический)*. Значение интервала от *–100* до *100*. На закладке *Limit Values (Уставки)* сконфигурирована *High Limit Value (Верхняя уставка)* со значением *55*. Для цветовой индикации выхода за уставку используется красный цвет. Окно закрывается нажатием на *OK*. 

Конфигурирование панели инструментов и строки статуса

Шаг	Действие: Конфигурирование панели инструментов и строки статуса
1	В *менеджере тегов* создайте два внутренних тега типа *Binary*. В данном примере это теги *BINi_ex_tlg_06* и *BINi_ex_tlg_07*.
2	Для управления процессом обновления нужно сконфигурировать объект *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*. В данном примере используется объект *Status Display3*. С помощью *конфигурационного диалога* объект связывается с тегом *BINi_ex_tlg_06*, его обновление активизируется изменением тега. Создаются состояния *0* и *1*. В данном примере этим состояниям назначены пиктограммы *stop tlg.bmp* и *stop go tlg.bmp*. Это окно закрывается нажатием на *OK*. У только что сконфигурированного объекта *Status Display3* создайте *процедуру Cи* для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Эта *процедура Cи* имитирует нажатие кнопки запуска/останова на стандартной панели инструментов. Статус тега *BINi_ex_tlg_06* инвертируется для отображения измененного состояния процесса обновления: нулевое значение тега соответствует активизированному обновлению. Значение тега *BINi_ex_tlg_06* равно нулю в момент открытия кадра, так как при этом автоматически запускается процесс обновления. Это реализовано с помощью *процедуры Cи* в пункте *Events (События)* → *Miscellaneous (Разное)* → *Open Picture (Открытие кадра)* объекта *ex_3_chapter_01b.pdl*. 
3	Сконфигурируйте второй объект окна статуса *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)* как показано в шаге 2. В данном примере это *Status Display2*. Этот объект отображает элементы управления для редактирования таблицы. Объект связан с тегом *BINi_ex_tlg_07*. Соответственно, используются другие пиктограммы (*Edit.gif* / *Edit inv.gif*). Создайте новую процедуру Cи для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*. Эта процедура имитирует нажатие кнопок редактирования на стандартной панели инструментов объекта. Состояние тега *BINi_ex_tlg_07* инвертируется для отображения измененного состояния таблицы: нулевое значение тега соответствует отключенной возможности редактирования. Значение тега *BINi_ex_tlg_07* всегда равно нулю в момент открытия кадра, так как при этом возможность редактирования таблицы отключается. Это реализовано с помощью *процедуры Cи* для события *Events (События)* → *Miscellaneous (Разное)* → *Open Picture (Открытие кадра)* объекта *ex_3_chapter_01b.pdl*. Она содержит строку, которая устанавливает значение тега в *0*. 

Шаг	Действие: Конфигурирование панели инструментов и строки статуса
4	Для навигации по архиву при остановленном процессе обновления используются четыре кнопки навигации стандартной панели инструментов. В данной реализации сконфигурированы четыре объекта *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*; это объекты *Button4*, *Button7*, *Button8* и *Button11*. У каждого из этих объектов для события *Events (События)* → *Mouse (Мышь)* → *Mouse Action (действия мыши)* создается *процедура Cu*. Эти процедуры имитируют нажатия кнопок на стандартной панели инструментов. Также требуется дополнительный объект *Smart Object (Интеллектуальный объект)* → *Graphic Object (Графический объект)*, который помещается поверх этих кнопок и делает их недоступными при включенном обновлении. В данном примере это объект *Graphic Object2*. Он отображает четыре отключенные кнопки (*Pfeile dis.bmp*). Для атрибута *Properties (Свойства)* → *Miscellaneous (Разное)* → *Display (Отображение)* создайте *Dynamic Dialog (Динамический диалог)*. Это окно контролирует видимость объекта в зависимости от тега *BINi_ex_tlg_06*, который содержит информацию об обновлении экранного элемента. 
5	Также требуется дополнительный графический объект *Smart Object (Интеллектуальный объект)* → *Graphic Object (Графический объект)*, который будет делать кнопку запуска/останова недоступной при включенном режиме редактирования. В данном примере для этого используется *Graphic Object1*. В пункте *Properties (Свойства)* → *Miscellaneous (Разное)* → *Display (Отображение)* создайте *динамический диалог*, который делает объект видимым, если значение тега *BINi_ex_tlg_07* равно *TRUE*, что позволяет редактировать таблицу. В качестве пиктограммы в данном примере используется *stop dis tlg.bmp*. Этот объект должен быть помещен точно поверх кнопки запуска/останова. 
6	Для строки статуса следует сконфигурировать *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере это *Button10*. Кнопки используются в качестве текстовых полей, так как им легко назначить 3D рамку, и, следовательно, не требуется использовать дополнительных элементов. Для *Button10* создайте *динамический диалог* у атрибута *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)*. Этот объект возвращает либо текст *Update Started (Обновление запущено)*, либо текст *Update Stopped (Обновление остановлено)*, в зависимости от значения тега *BINi_ex_tlg_06*. 

Процедура Си для WinCC Online Table Control (Control1)

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{

if  (GetTagDouble("G64i_ex_tlg_04") >= 55)
      {
      SetTagDouble("G64i_exTlg_08",GetTagDouble("G64i_ex_tlg_04"));
      }

return 50;
}
```

Считывание тега G64I_ex_tlg_04 и проверка, не превосходит ли он 55.

Если его значение больше 55, оно записывается в тег D64I_ex_tlg_08.

Процедура Си для кнопки редактирования (Status Display2)

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszl
{

TlgTableWindowPressEditRecordButton("AZ_ProcessValueArchive_00");

SetTagBit("BINi_ex_tlg_07",(SHORT)!GetTagBit("BINi_ex_tlg_07"));

SetTagBit("BINi_ex_tlg_06",TRUE);

}
```

Вызов стандартной функции *TlgTableWindowPressEditButton* эквивалентен нажатию кнопки редактирования на стандартной панели инструментов. Текст, назначенный функции, позволяет идентифицировать объект. Этот текст представляет собой имя окна, которое было задано при конфигурировании, в данном случае это строка *AZ_ProcessValueArchive_00*.

Инвертирование тега *BINi_ex_tlg_07* для сохранения текущего статуса режима редактирования.

Установка в TRUE тега BINi_ex_tlg_06 для остановки процесса обновления.

Процедура Си для навигационной кнопки запуска (Button4)

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

TlgTableWindowPressFirstButton("AZ_ProcessValueArchive_00");

}
```

Вызов этой стандартной функции эквивалентен нажатию кнопки *First Data Record (Первая запись)* стандартной панели инструментов. Другими кнопками используются следующие функции:

- *TlgTrendWindowPressPrevButton*
- *TlgTrendWindowPressNextButton*
- *TlgTrendWindowPressLastButton*
- *TlgTableWindowPressOpenTimeSelectDlgButton*
- *TlgTableWindowPressStartStopButton*

Замечание:

Для каждой кнопки стандартной панели инструментов объекта *WinCC Online Table Control* существует соответствующая стандартная функция, которая имитирует ее нажатие.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

Архивируемые теги должны быть настроены соответствующим образом.

Пользователем должно быть определено событие, которое будет инициировать архивацию. Для этого должна быть создана функция проекта.

Внешний вид элементов может быть настроен в соответствии с вашими требованиями. Это относится и к строке статуса.

4.1.4 Определяемый пользователем формат таблицы (ex_3_chapter_01c.pdl)

Постановка задачи

Непрерывно опрашивать значение процесса. За каждый 10–секундный период должны быть замерены среднее, максимальное и минимальное значения, которые затем должны сохраняться во внутреннем архиве проекта.
Сохраненные значения должны отображаться в виде таблицы, созданной пользователем в *графическом дизайнере*. Определенный пользователем формат таблицы необходим, когда не подходят стандартные средства подсистемы *Tag Logging*.

Концепция реализации

Для архивирования данных в редакторе *Tag Logging* создается непрерывный циклический архив значений процесса.
Для реализации графического объекта используются *Standard Object (Стандартный объект)* → *Static Text (Статический текст)* или *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*.
Данные считываются из таблицы базы данных соответствующего архива с помощью API функций.

Создание архива значений процесса

Шаг	Действие: Создание архива значений процесса
1	В *менеджере тегов* создайте три тега типа *Floating–Point Number 64–Bit IEEE 754 (64–битное число с плавающей точкой IEEE 754)*. В данном примере это теги *G64i_ex_tlg_05*, *G64i_ex_tlg_06* и *G64i_ex_tlg_07*. В эти теги будут записываться архивные значения.
2	Создайте новый *архив значений процесса*, используя Archive Wizard. В данном примере этот архив назван *ZK_ProcessValueArchive_02*. В качестве архивируемого тега был трижды выбран тег *G64i_ex_tlg_04*.
3	В окне свойств архива значений процесса установлена длина архива — *100* записей. Для остальных опций оставлены значения по умолчанию.

Шаг	Действие: Создание архива значений процесса
4	В окне свойств первого тега процесса на закладке *General Information (Общая информация)* качестве *Name of the Archive Tag (Имени архивного тега)* введите *MaximumValue*. В поле *Also write Archived Value to Tag (Также записывать в тег архивное значение)* выбран тег *G64i_ex_tlg_07*. С этим тегом может взаимодействовать процедура Си, которая будет реагировать на архивирование значения. Это реализуется активизацией процедуры Си по изменению тега (upon change). На закладке *Parameters (Параметры)* значения *Acquisition (Опроса)* установлены следующим образом: *Cycle (Цикл)* – *500 ms*, *Archiving (Архивирование)* – *20*500 ms*. В поле *Processing (Обработка)* выбран вариант *Maximum Value (Максимальное значение)*. Это означает, что выбранный тег опрашивается каждые 500 мс и архивируется каждые 10 с. При этом архивируется наибольшее значение за указанный период (10 секунд). Для остальных опций оставлены значения по умолчанию. 
5	Аналогичным образом конфигурируются два оставшихся тега. При этом в поле *Processing* выбираются варианты *Minimum Value (Минимальное значение)* и *Mean Value (Среднее значение)*. Соответствующим образом выбираются имена этих архивных тегов.

Реализация в графическом дизайнере

Шаг	Действие: Реализация в графическом дизайнере
1	В *менеджере тегов* создайте два внутренних тега типа *Binary (Двоичный)*. В данном примере это теги *BINi_ex_tlg_06* и *FLAG_TableGetOutputValue*.
2	Создание функции проекта, с помощью которой подсистема *Tag Logging* передаст архивные данные в другую функцию (используется callback function – функция обратного вызова). Эта функция вызывается один раз для каждой записи и содержит информацию об этой записи в специальной структуре. Переданные данные сохраняются в статическом массиве таких структур. В данном примере используются функции *EnumerateSuperArchiveData* и *GetArchiveDataCallback*. Пример использует две внешние переменные Си. extern int dwSize extern WORD wOffset Они должны быть созданы при запуске проекта; для этого используется отдельная функция проекта. Ее вызов вставляется в *процедуру Си* для события *Events (События)* → *Miscellaneous (Разное)* → *Open Picture (Открытие кадра)* начального кадра *ex_0_startpicture_00.PDL*. В данном примере эта функция названа *CreateExternal*.

Шаг	Действие: Реализация в графическом дизайнере
3	Создайте новый кадр; в примере это кадр *ex_3_chapter_01b.pd*. Укажите, что должно отображаться 10 строк. Для отображения данных в первом столбце используется 10 объектов *Standard Objects (Стандартные объекты)* → *Static Texts (Статический текст)*, которые выводят дату и время. Для дополнительных столбцов используются объекты *Smart Objects (Интеллектуальные объекты)* → *I/O Fields (Поля ввода/вывода)*. В качестве имен объектов в первом столбце используется текст объектов с *Static Text1* по *Static Text10*, где номер определяет строку. Нумерация осуществляется снизу вверх, так как последняя строка содержит самые новые данные. *Поля ввода/вывода* используют в качестве имени числовой код. Первая цифра соответствует номеру столбца, вторая — номеру строки
4	У каждого *статического текста* для атрибута *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* создается *процедура Cu*. Эта процедура считывает данные для отображения из функции обратного вызова, используя номер своего объекта. Эта функция активизируется при изменении тега *FLAG_TableGetOutputValue*. Состояние данного тега меняется, когда в архив записываются новые данные или эти данные из него считываются. Точно таким же образом создайте *процедуру Cu* для каждого *поля ввода/вывода* в *Properties (Свойства)* → *Output/Input (Вывод/Ввод)* → *Output Value (Выводимое значение)*. Эта процедура служит также для чтения из возвратной функции записи, назначенной объекту.
5	Для управления обновлением сконфигурируйте объект *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*. В данном примере используется объект *Status Display3*. С помощью *конфигурационного диалога* объект соединяется с тегом *BINi_ex_tlg_06* и активизируется при его изменении. Создаются состояния *0* и *1*, им назначаются соответствующие пиктограммы. В данном примере используются пиктограммы *stop go tlg.gif* и *stop tlg.gif*. 

Date/Time	Mean Value	Minimum	Maximum
06.07.1999 10:17:27	000,00	000,00	000,00
06.07.1999 10:17:37	000,00	000,00	000,00
06.07.1999 10:17:47	000,00	000,00	000,00
06.07.1999 10:17:57	000,00	000,00	000,00
06.07.1999 10:18:07	000,00	000,00	000,00
06.07.1999 10:18:17	000,00	000,00	000,00
06.07.1999 10:18:27	000,00	000,00	000,00
06.07.1999 10:18:37	000,00	000,00	000,00
06.07.1999 10:18:47	000,00	000,00	000,00
06.07.1999 10:18:57	000,00	000,00	000,00

Шаг	Действие: Реализация в графическом дизайнере
6	Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте *процедуру Си*, которая будет инвертировать состояние тега *BINi_ex_tlg_06*. Состояние *TRUE (Истинно)* означает, что процесс обновления запущен. Значение этого тега всегда равно *TRUE (Истинно)* в момент открытия кадра, так как при этом автоматически запускается процесс обновления окна таблицы. Это реализуется с помощью процедуры Си для события *Events (События)* → *Miscellaneous (Разное)* → *Open Picture (Открытие кадра)* объекта *ex_3_chapter_01c.pdl*. Эта процедура Си устанавливает значение тега в *TRUE (Истинно)* и один раз считывает значения.
7	Для объекта *Statusdisplay3* в *Properties (Свойства)* → *Geometry (Геометрия)* → *Width (Ширина)* создайте процедуру Си. Эта процедура в зависимости от состояния тега *BINi_ex_tlg_06* читает архив и инвертирует тег *FLAG_TableGetOutputValue* для того чтобы активизировать обновление таблицы. Эта процедура Си запускается при изменении тега *G64i_ex_tlg_07*, где также хранится архивное значение. Эта процедура также действует при следующей архивации, если записываемое значение идентично предыдущему. Для этого в тег *G64i_ex_tlg_07* записывается значение, заведомо недостижимое тегом процесса, оно будет архивироваться после каждого запуска данной *процедуры Си*.
8	Когда обновление не производится, для навигации по архиву используются навигационные кнопки. В данной реализации используются четыре кнопки *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*, в данном случае это объекты *Button4*, *Button7*, *Button8* и *Button11*. 
9	Для события *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* создаются *процедуры Си*. Эти процедуры записывают новое значение во внешнюю переменную Си *dwOffset*. Тег *FLAG_TableGetOutputValue* инвертируется для контроля обновления объекта. Дополнительно требуется *Smart Object (Интеллектуальный объект)* → *Graphic Object (Графический объект)*, который помещается поверх этих кнопок и делает их недоступными при включенном обновлении. В данном примере это объект *Graphic Object2*. Он отображает четыре отключенные кнопки (*Pfeile dis.bmp*). Для атрибута *Properties (Свойства)* → *Miscellaneous (Разное)* → *Display (Отображение)* создайте *соединение с тегом BINi_ex_tlg_06*.

Функция проекта для чтения из архива

```
#include "apdefap.h"
BOOL EnumerateSuperArchiveData()
{

extern DWORD   dwSize;
BOOL           fRet;
TLG_GETARCHIVDATA      GAD;
CMN_ERROR      Error;
LPTSTR  lpszArchivName = "ZK_ProcessValueArchive_02";
LPTSTR  lpszVarName3 = "MaximumValue" ;
LPTSTR  lpszVarName2 = "MinimumValue" ;
LPTSTR  lpszVarName1 = "MeanValue" ;
SYSTEMTIME     sysFrom;
SYSTEMTIME     sysTo;
time_t  Time;
struct tm*     TimeStruct;

time(&Time);
TimeStruct = localtime(&Time);

sysTo.wYear    = (WORD)(TimeStruct->tm_year+1900);
sysTo.wMonth   = (WORD)(TimeStruct->tm_mon+1);
sysTo.wDay     = (WORD)(TimeStruct->tm_mday);
sysTo.wHour    = (WORD)(TimeStruct->tm_hour);
sysTo.wMinute  = (WORD)(TimeStruct->tm_min);
sysTo.wSecond  = (WORD)(TimeStruct->tm_sec);

sysFrom.wYear    = 1997;
sysFrom.wMonth   = 1;
sysFrom.wDay     = 1;
sysFrom.wHour    = 0;
sysFrom.wMinute  = 0;
sysFrom.wSecond  = 0;

Call(&GAD,(PVOID)0);

if (TLGConnect( NULL , &Error )==FALSE) {
        printf("Error: %s\r\n",Error.szErrorText);
        return FALSE;
        }
else {
        fRet=TLGGetArchivData(lpszArchivName,lpszVarName1,
                sysFrom,sysTo,GetArchiveDataCallback,
                (PVOID)1,0,&Error);
        if (fRet==FALSE)
                printf("Error: %s\r\n",Error.szErrorText);

        fRet=TLGGetArchivData(lpszArchivName,lpszVarName2,
                sysFrom,sysTo,GetArchiveDataCallback,
                (PVOID)2,0,&Error);
        if (fRet==FALSE)
                printf("Error: %s\r\n",Error.szErrorText);

        fRet=TLGGetArchivData(lpszArchivName,lpszVarName3,
                sysFrom,sysTo,GetArchiveDataCallback,
                (PVOID)3,0,&Error);
        if (fRet==FALSE)
                printf("Error: %s\r\n",Error.szErrorText);

        Call(&GAD,(PVOID)4);
        dwSize=GAD.dwFlags;
        TLGDisconnect( NULL );
        return TRUE;
        }
}
```

Определение времени запуска и останова процесса чтения данных из архива. В качестве начального времени берется фиксированное время, в качестве конечного — текущее время системы.

Инициализация функции обратного вызова с помощью функции *Call*. Эта функция вызывает функцию *GetArchiveDataCallback* со значением 0 в качестве параметра *lpUser*.

Установка соединения с подсистемой *Tag Logging*. Если это не удается, функция прекращает работу.

Чтение значений из архивных тегов *MaxValue*, *MinValue* и *MeanValue* с помощью функции *TLGGetArchiveData*.

Определение количества считанных значений. Это делается с помощью функции *Call*, которая вызывает функцию *GetArchiveDataCallback* со значением 4 в качестве параметра *lpUser*.

Разрыв соединения с подсистемой *Tag Logging*.

Функция обратного вызова

```
BOOL GetArchiveDataCallback (PTLG_GETARCHIVDATA      lpGAD, PVOID   lpUser)
{

static int i1 = 0;
static int i2 = 0;
static int i3 = 0;

WORD wRecordNumber;
WORD wColumnNumber;

static TLG_GETARCHIVDATA GAD1[100];
static TLG_GETARCHIVDATA GAD2[100];
static TLG_GETARCHIVDATA GAD3[100];

int User;
User=(int)lpUser;

if ((User==1)||(User==2)||(User==3))
{

switch(User)
{
case 1 : if (i1<=100) {GAD1[i1]=*lpGAD; i1++;} break;
case 2 : if (i2<=100) {GAD2[i2]=*lpGAD; i2++;} break;
case 3 : if (i3<=100) {GAD3[i3]=*lpGAD; i3++;} break;
}//switch

}//if ((User==1)||(User==2)||(User==3))

if (User==0)
{

i1=0;
i2=0;
i3=0;

memset(&GAD1[0],0,sizeof(TLG_GETARCHIVDATA)*100);
memset(&GAD2[0],0,sizeof(TLG_GETARCHIVDATA)*100);
memset(&GAD3[0],0,sizeof(TLG_GETARCHIVDATA)*100);

}//if (User==0)

if (User==4)
{
lpGAD->dwFlags=i1;
}//if (User==4)

if (User==7)
{

wRecordNumber=lpGAD->stTime.wMonth;
wColumnNumber=lpGAD->stTime.wDay;

switch(wColumnNumber)
{
case 0   : lpGAD->stTime.wYear = GAD1[wRecordNumber].stTime.wYear;
           lpGAD->stTime.wMonth = GAD1[wRecordNumber].stTime.wMonth;
           lpGAD->stTime.wDay = GAD1[wRecordNumber].stTime.wDay;
           lpGAD->stTime.wHour = GAD1[wRecordNumber].stTime.wHour;
           lpGAD->stTime.wMinute = GAD1[wRecordNumber].stTime.wMinute;
           lpGAD->stTime.wSecond = GAD1[wRecordNumber].stTime.wSecond;
           break;
case 1   : lpGAD->doValue=GAD1[wRecordNumber].doValue;
           break;
case 2   : lpGAD->doValue=GAD2[wRecordNumber].doValue;
           break;
case 3   : lpGAD->doValue=GAD3[wRecordNumber].doValue;
           break;
default  : break;
}//switch

}//if (User==7)

return TRUE;
}
```

Объявление трех статических счетчиков тегов: *i1*, *i2* и *i3*.

Объявление трех статических массивов, состоящих из структур типа TLG_GETARCHIVEDATA. В данных массивах хранятся архивные значения.

Если параметр *lpUser* равен 1, 2 или 3, функция была вызвана подсистемой *Tag Logging*. В этом случае переданная структура *lpGAD* сохраняется в соответствующем массиве.

Если параметр *lpUser* равен 0, то производится инициализация. Счетчики тегов сбрасываются в 0, массивы очищаются.

Если параметр *lpUser* равен 4, то запрашивается число сохраненных значений, которое сохраняется в переданной структуре в поле *dwFlags*.

Если параметр *lpUser* равен 7, запрашивается значение тега в объектах *I/O Field (Поле ввода/вывода)* или *Static Text (Статический текст)*. Позиция таблицы, в которой находится запрашиваемый тег, определяется полями *stTime.wMonth* и *stTime.wDay* переданной структуры.

Процедура Си для статических текстов

```
#include "apdefap.h"
 char* _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

int nRecordNumber;
WORD wColumnNumber;
extern WORD wOffset;
char szObject[5];
TLG_GETARCHIVDATA GAD;
PVOID lpUser7 = 7;
char szTime[20] = "";
int nObjectNumber;
extern DWORD dwSize;

wColumnNumber=0;

nObjectNumber=atoi(lpszObjectName+15);
nRecordNumber=(dwSize-nObjectNumber-wOffset);

if (nRecordNumber<0) return "";

GAD.stTime.wMonth=(WORD)nRecordNumber;
GAD.stTime.wDay=wColumnNumber;

GetArchiveDataCallback(&GAD,lpUser7);

sprintf(szTime,"%02d.%02d.%d %02d:%02d:%02d",
GAD.stTime.wDay,GAD.stTime.wMonth,GAD.stTime.wYear,GAD.stTime.wHour,
GAD.stTime.wMinute,GAD.stTime.wSecond);

return szTime;

}
```

В поля *stTime.wMonth* и *stTime.wDay* передаваемой структуры *GAD* записываются номера столбца или вычисленный номер записи. Информацию о номере записи содержит имя объекта.

Функция *GetArchiveDataCallback* вызывается, если значение параметра *lpUser* равно 7, т.е. когда запрашивается значение.

Значение сохраняется в поле *stTime* передаваемой структуры *GAD*. В соответствии с этим формируется отображаемый текст, который возвращается посредством оператора *return*.

Процедура Си для полей ввода/вывода

```
#include "apdefap.h"
 double _main(char* lpszPictureName, char* lpszObjectName, char* lpszProper
{

extern dwSize;
int nRecordNumber;
WORD wColumnNumber;
extern WORD wOffset;
char szObject[5];
TLG_GETARCHIVDATA GAD;
PVOID lpUser = 7;

strcpy(szObject,"");
sprintf(szObject,"%c",lpszObjectName[0]);
wColumnNumber=(WORD)atoi(szObject);

nRecordNumber=(dwSize-atoi(lpszObjectName+1)-wOffset);

if (nRecordNumber<0) return 0;

GAD.stTime.wMonth=(WORD)nRecordNumber;
GAD.stTime.wDay=wColumnNumber;

GetArchiveDataCallback(&GAD,lpUser);

return GAD.doValue;

}
```

По имени объекта определяется номера строки и столбца. В поля *stTime.wMonth* и *stTime.wDay* передаваемой структуры *GAD* записываются номер столбца или вычисленный номер записи.

Функция *GetArchiveDataCallback* вызывается, если значение параметра *lpUser* равно 7, т.е. когда запрашивается значение тега.

Значение тега сохраняется в поле *doValue* передаваемой структуры *GAD* и используется в качестве возвращаемого значения.

Замечание:

Панель инструментов содержит кнопку, изображенную ниже, которая служит для изменения свойств таблицы. С помощью этой кнопки вызывается окно выбора цвета различных элементов таблицы. Короткое описание данного окна можно найти в примере Color Dialogs (Диалоги выбора цвета) (ex_3_chapter_01c).

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

Отображаемые архивные данные должны быть настроены соответствующим образом.

Настройте формат таблицы в соответствии с вашими требованиями. Если необходимы дополнительные строки или столбцы, соответствующим образом должны быть изменены процедуры Си и функции проекта.

4.1.5 Архивация двоичных тегов (ex_3_chapter_01d.pdl)

Постановка задачи

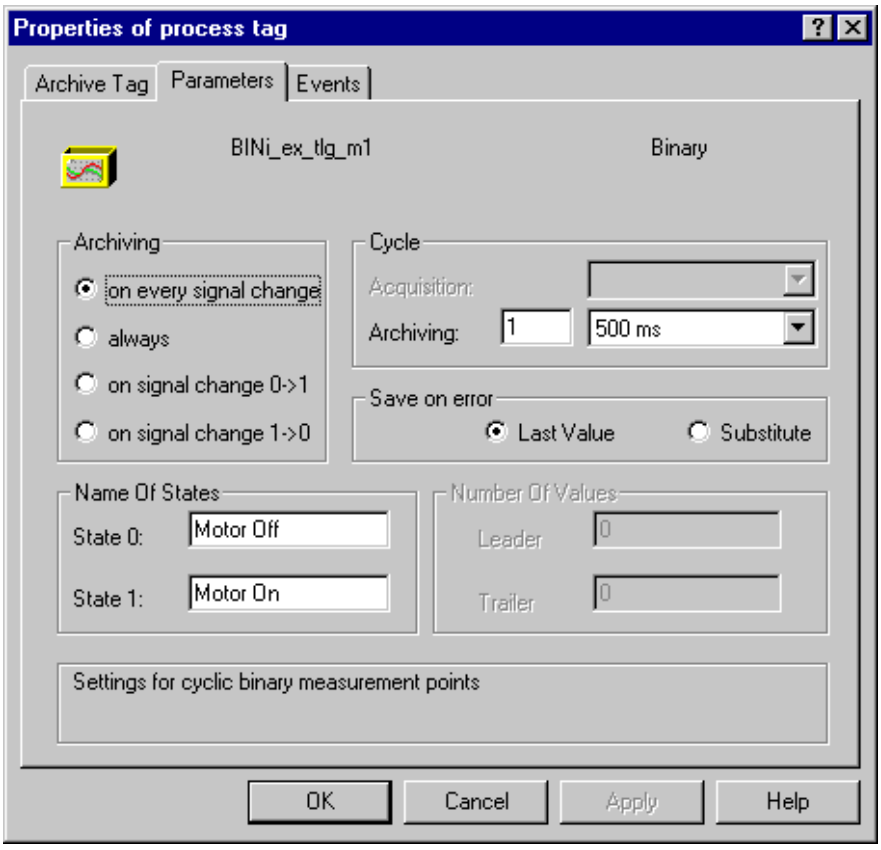
Фиксировать в архиве операции включения трех моторов. При выборе мотора с помощью (мыши) должна выводится таблица, содержащая операции включения за последние сутки. Таблица должна отображать состояния только выбранного мотора.

Концепция реализации

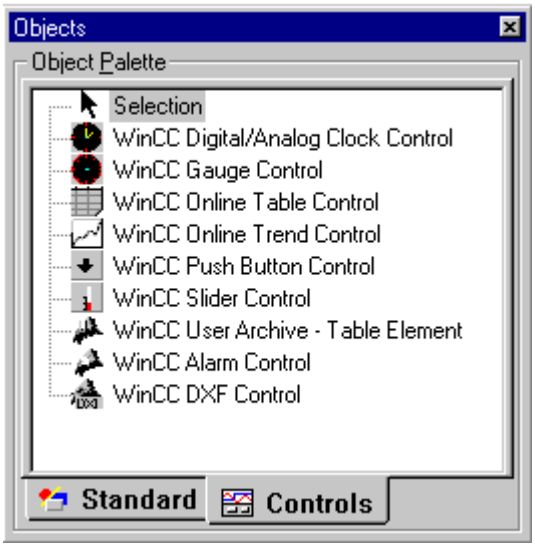
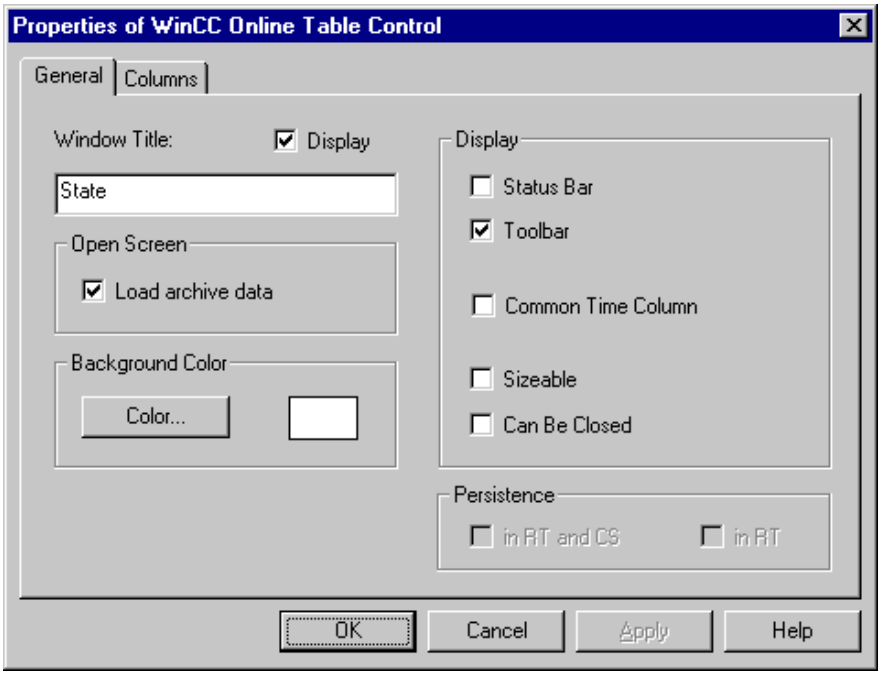
Для архивирования отображаемых данных в редакторе *Tag Logging* создается непрерывный циклический архив. Каждый тег отображается в отдельной таблице.
Для реализации графического объекта, отображающего нужные столбцы в зависимости от выбранного мотора, используется объект *WinCC Online Table Control*.

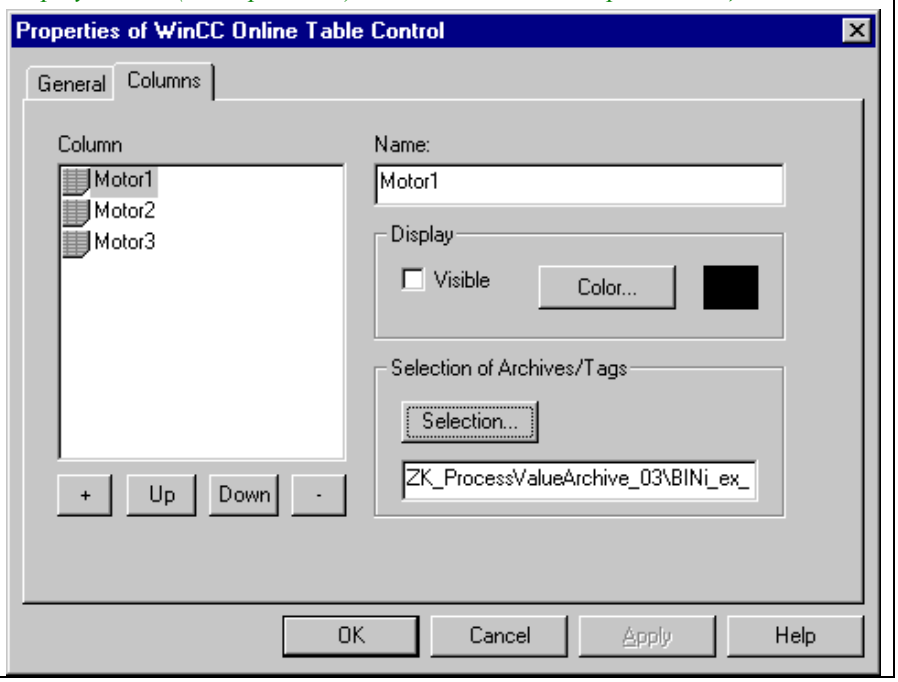
Создание архива значений процесса

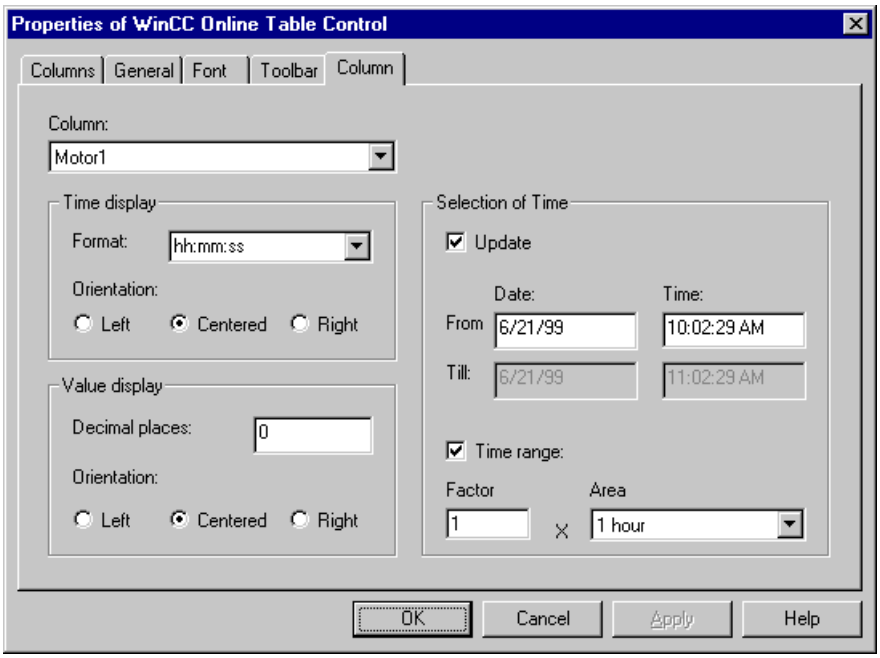
Шаг	Действие: Создание архива значений процесса
1	В *менеджере тегов* создайте четыре тега. В данном примере это двоичные теги *BINi_ex_tlg_m1*, *BINi_ex_tlg_m2* и *BINi_ex_tlg_m3* и тег *U08i_ex_tlg_00* типа *Unsigned 8–Bit Value (8–битная величина без знака)*.
2	С помощью мастера архивов создайте новый *архив значений процесса*. В данном примере это архив *ZK_ProcessValueArchive_03*. Для архивирования выберите теги *BINi_ex_tlg_m1*, *BINi_ex_tlg_m2* и *BINi_ex_tlg_m3*.
3	В окне свойств архива значений процесса установите размер архива — *40* записей. Для остальных опций оставлены значения по умолчанию.

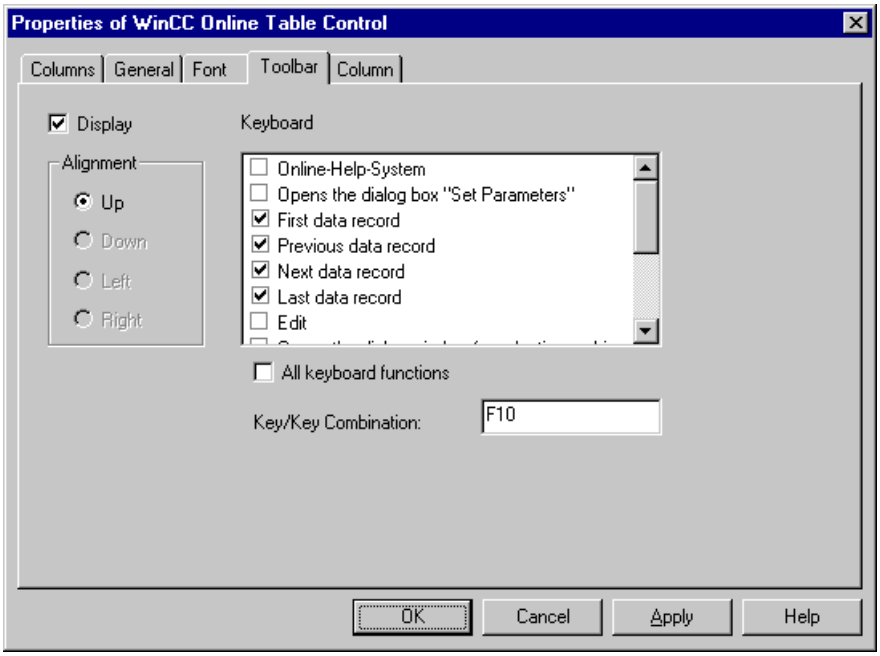
Шаг	Действие: Создание архива значений процесса
4	В окне свойств первого тега процесса в поле *Archive upon (Архивирование по…)* закладки *Parameters (Параметры)* выберите вариант *Every Signal Change (По каждому изменению сигнала)*. В поле *Name of the Status (Имя состояния)* введите *Motor Off (Мотор выключен)* для состояния *Status 0* и *Motor On (Мотор включен)* для *Status 1*. В поле *Cycle (Цикл)* выберите *Archiving (Архивирование) 1*500 ms*. Для остальных опций оставлены значения по умолчанию. 

Конфигурирование окна таблицы

Шаг	Действие: Конфигурирование окна таблицы
1	Создайте новый кадр, в данном примере это кадр *ex_3_chapter_01d.pdl*.
2	Конфигурирование объекта, используемого для отображения таблицы. Это *WinCC Online Table Control*. Он выбирается из *Object Palette (Палитры объектов)* и затем помещается в кадр. 
3	После добавления *WinCC Online Table Control* в кадр автоматически открывается окно *Properties (Свойства)*. На закладке *General Information (Общая информация)* вы можете определить название объекта и назначенный ему текст. В качестве *Background Color (Цвета фона)* выбран белый цвет. В области *Display (Отображение)* опции *Sizeable (Изменение размера)* и *Toolbar (Панель инструментов)* отключены. 

Шаг	Действие: Конфигурирование окна таблицы
4	На закладке *Columns (Столбцы)* выполняются настройки отображаемых столбцов. Один столбец уже создан. Переименуйте его в *Motor1*. С помощью кнопки *Selection (Выбор)* столбцу назначается архивный тег. В данном примере столбцу назначается тег *BINi_ex_tlg_m1* уже созданного архива *ZK_ProcessValueArchive_03*. Добавьте еще два столбца и назначьте им теги *BINi_ex_tlg_m2* и *BINi_ex_tlg_m3*; введите имена столбцов и выберите для них цвет. Опция *Display Visible (Отображать)* отключена *для всех трех столбцов*. 

Шаг	Действие: Конфигурирование окна таблицы
5	Индивидуальная настройка столбцов. Для этого используется расширенное диалоговое окно, которое открывается D (двойным щелчком мыши) на элементе управления. Окно свойств, описанное выше, можно открыть с помощью D (двойного щелчка мыши) и клавиши CTRL. Расширенное окно свойств в дополнение к уже описанным закладкам *General Information (Общая информация)* и *Columns (Столбцы)* содержит еще три. На закладке *Column (Столбец)* производятся следующие изменения: *Format (Формат)* объекта *Time Display (Индикатор времени)* установлен в *hh:mm:ss*. В качестве *Orientation (Ориентации)* выбран вариант *Centered (По центру)*, десятичные позиции заполняются нулями. В поле *Time Selection (Выбор времени)* опция *Time Range (Период времени)* включена, но *Range (Период)* установлен в *1 X 1 Час*. 

Шаг	Действие: Конфигурирование окна таблицы
6	На закладке *Toolbar (Панель инструментов)* в области *Key Functions (Функции кнопок)* включаются следующие опции: First Data Record (Первая запись) Previous Data Record (Предыдущая запись) Next Data Record (Следующая запись) Last Data Record (Последняя запись) Select Time Range (Выбор периода времени) Start/Stop the Update (Запуск/останов обновления) 
7	Так как отображается только один столбец, в качестве размера шрифта на закладке *Font (Шрифт)* для улучшения читаемости выбрано значение 13.5 пунктов. Для остальных опций оставлены значения по умолчанию. Окно закрывается нажатием на *OK*.

Реализация в графическом дизайнере

Шаг	Действие: Реализация в графическом дизайнере
1	Изображения моторов состоят из объектов *Standard Object (Стандартный объект)* → *Circle (Окружность)*, *Standard Object (Стандартный объект)* → *Polygon (Многоугольник)* и *Standard Object (Стандартный объект)* → *Static Text (Статический текст)*. Цвет фона окружности изменяется в зависимости от состояния мотора с помощью *динамического диалога*. Указанные три объекта должны быть сгруппированы. В данном примере это группы *Group1*, *Group2* и *Group3*. У каждого из этих объектов установлено *прямое соединение* для события *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)*, которое записывает номер мотора в тег *U08i_ex_tlg_00*, и создана *процедура Си* для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)*, которая делает видимым текущий столбец (или невидимыми другие).
2	Каждому мотору назначены две *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*. Эти кнопки управляют тегами отдельных моторов посредством *прямого соединения*.
3	Для идентификации выбранного в данный момент мотора, каждому из них назначается объект *Standard Object (Стандартный объект)* → *Rectangle (Прямоугольник)*. В поле *Properties (Свойства)* → *Styles (Стили)* → *Line Style (Стиль линии)* выбран пунктирный стиль, а в *Properties (Свойства)* → *Styles (Стили)* → *Fill Pattern (Узор)* – *Transparent (Прозрачный)*. Для поля *Properties (Свойства)* → *Miscellaneous (Разное)* → *Display (Отображение)* создан *динамический диалог*, который делает объект *Rectangle (Прямоугольник)* видимым только когда содержимое тега *U08i_ex_tlg_00* соответствует собственному номеру объекта.

Процедура Си для Motor1 (Group 1)

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpsz
{


//show column 1
SetPropWord(lpszPictureName,"Control1","Index",0);
SetPropBOOL(lpszPictureName,"Control1","ItemVisible",TRUE);


//hide column 2
SetPropWord(lpszPictureName,"Control1","Index",1);
SetPropBOOL(lpszPictureName,"Control1","ItemVisible",FALSE);

//hide column 3
SetPropWord(lpszPictureName,"Control1","Index",2);
SetPropBOOL(lpszPictureName,"Control1","ItemVisible",FALSE);

}
```

С помощью функции *SetPropWord* нулевой индекс ассоциируется с объектом *Control1*. Это соответствует первому столбцу. С помощью *SetPropBOOL* этот столбец делается видимым.

Те же действия выполняются для выключения видимости остальных столбцов.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

Архивные теги должны быть настроены соответствующим образом.

Внешний вид объектов может быть настроен в соответствии с вашими требованиями.

4.1.6 Архивация в определенные моменты времени (ex_3_chapter_01e.pdl)

Постановка задачи

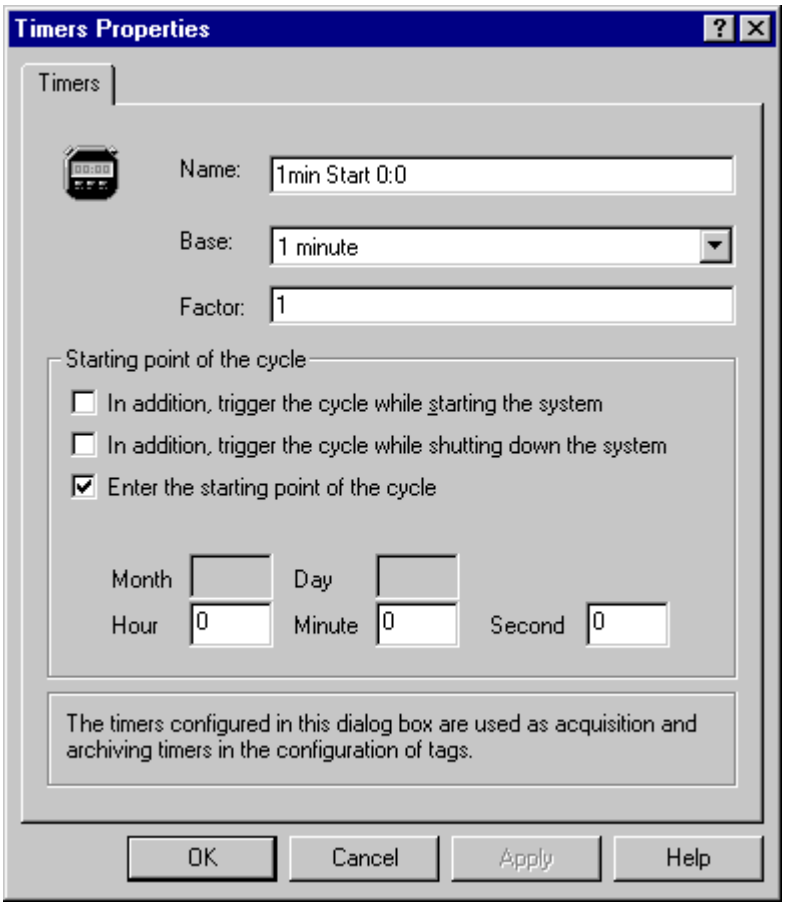
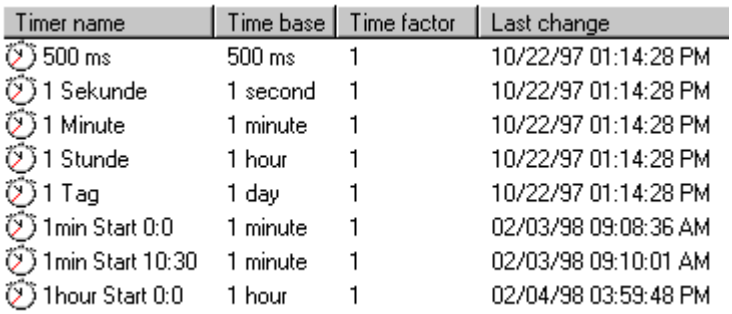
Использовать непрерывный циклический архив значений процесса для опроса значений процесса с периодом 1 с. Каждую минуту должна архивироваться сумма данных значений.
Архивные значения должны отображаться в виде таблицы; посредством стандартных средств должны быть реализованы панель инструментов и строка статуса.

Концепция реализации

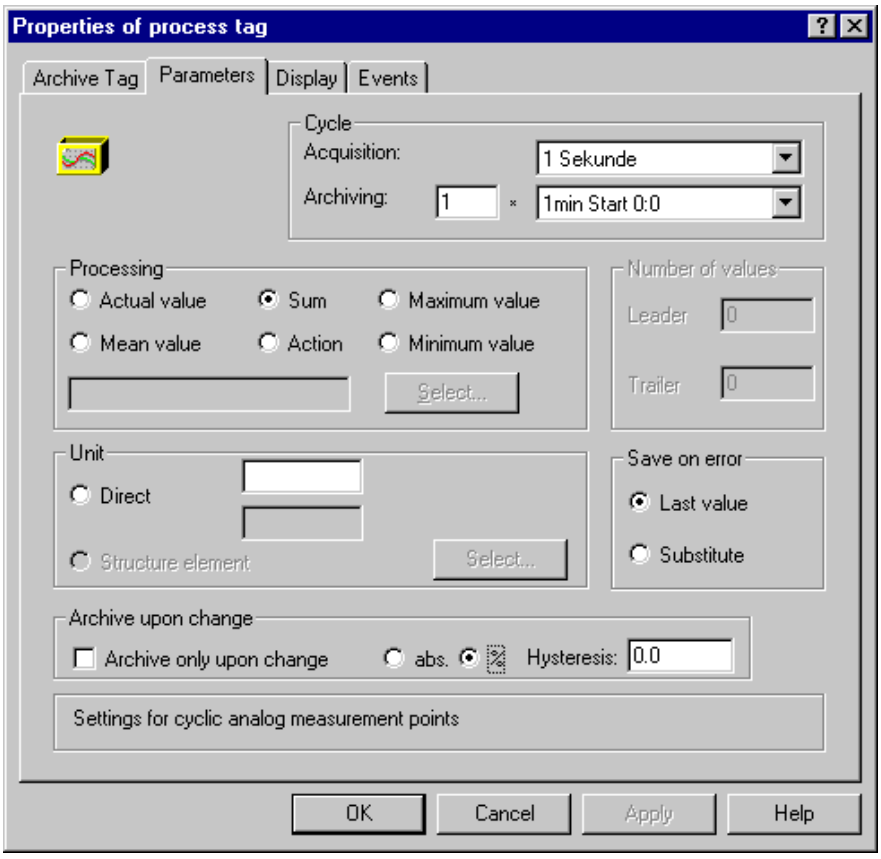
Для архивирования отображаемых данных в редакторе *Tag Logging* создается непрерывный циклический архив значений процесса. Для поминутной архивации используется объект *Timer (Таймер)*. Данный таймер запускается в определенное время, с которого начинается архивация.
Для отображения данных используется объект *WinCC Online Table Control*.

Создание нового таймера

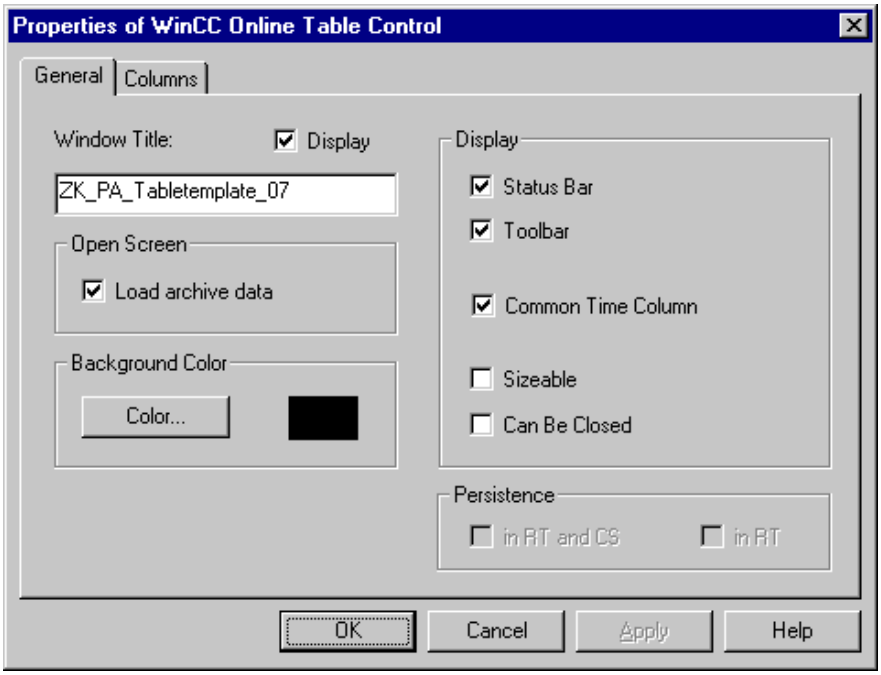
Шаг	Действие: Создание нового таймера
1	В окне *проводника WinCC* откройте редактор *Tag Logging*. Создайте новый *таймер*, нажав R (правой кнопкой мыши) на соответствующем поле навигационного окна 

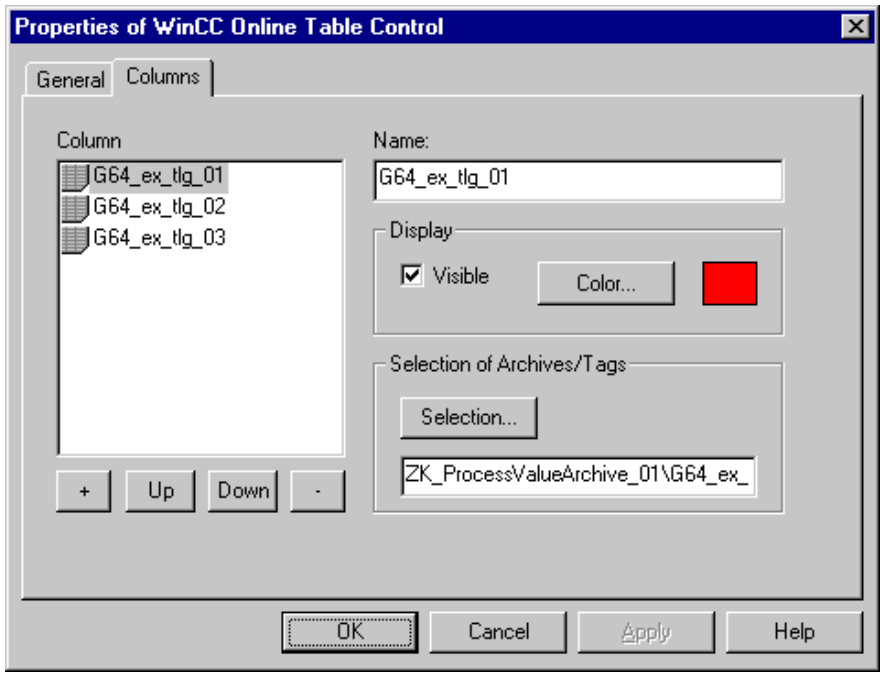
Шаг	Действие: Создание нового таймера
2	Откроется окно свойств таймера. В качестве *Name (Имени)* таймера в данном примере выбрана строка *1min Start 0:0*. *Base (База) = 1 Minute (1 минута)*, *Factor (Множитель) = 1*. Множитель позволяет создавать, например, четырех– или шестиминутные таймеры. В поле *Starting Point of the Cycle (Начало цикла)* выберите вариант *Enter the Starting Point of the Cycle (Значение задано пользователем)*. В каждом поле введите *0*, в этом случае цикл будет активизироваться по прошествии каждой полной минуты. Если бы было введено конкретное время, цикл запускался бы один раз при достижении установленного времени. Закройте окно, нажав на *OK*. 
3	В правом окне вместе с уже установленными по умолчанию таймерами появится только что созданный. 

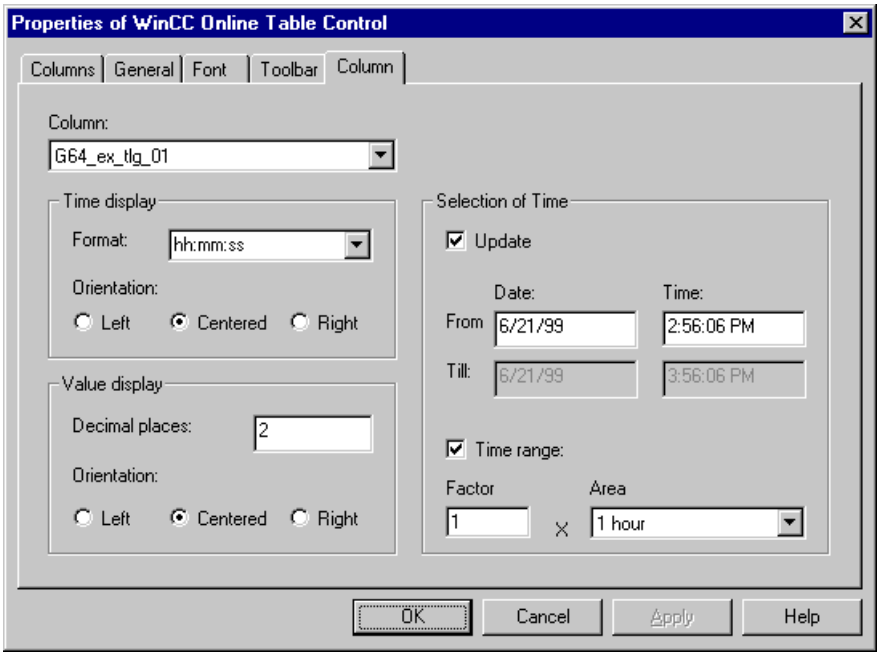
Создание архива значений процесса

Шаг	Действие: Создание архива значений процесса
1	С помощью мастера архивов создайте новый архив значений процесса. В данном примере архив назван *ZK_ProcessValueArchive_01*. В качестве тегов для архивации выберите теги *G64_ex_tlg_01*, *G64_ex_tlg_02* и *G64_ex_tlg_03*. В данном примере их значения поставляются имитатором. В окне свойств архива оставьте настройки, произведенные мастером.
2	В окне свойств первого тега процесса в поле *Cycle (Цикл)* закладки *Parameters (Параметры)* выберите значение *1 Second (1 секунда)*. В поле *Archiving (Архивация)* укажите *1* * *1min Start 0:0*. В поле *Processing (Обработка)* выберите вариант *Sum (Сумма)*. Для оставшихся свойств первого тега оставьте значения по умолчанию. Выполните эту же последовательность действий для остальных архивных тегов. 

Реализация в графическом дизайнере

Шаг	Действие: Конфигурирование окна таблицы
1	Создайте новый кадр; в примере это кадр *ex_3_chapter_01e.pdl*.
2	Конфигурирование объекта, используемого для отображения таблицы – *WinCC Online Table Control*. Он выбирается из *Object Palette (Палитры объектов)* и помещается в кадр.
3	После добавления объекта в кадр автоматически открывается окно свойств *WinCC Online Table Control*. На закладке *General Information (Общая информация)* введите имя объекта. В данном примере в качестве имени указано *ZK_ProcessValueArchive_01*. С помощью кнопки *Color (Цвет)* выбран *Background Color (Цвет фона)*, в данном случае черный. В области *Display (Отображение)* включены опции *Sizeable (Изменение размера)* и *Shared Time Column (Общий столбец времени)*. 

Шаг	Действие: Конфигурирование окна таблицы
4	На закладке *Columns (Столбцы)* производится настройка столбцов. В данном примере нам нужно создать три столбца. Один столбец уже создан и переименован в *G64_ex_tlg_01*. С помощью кнопки *Selection (Выбор)* столбцу назначается *архивный тег*. В данном примере для этого используется тег *G64_ex_tlg_01* из уже созданного архива *ZK_ProcessValueArchive_0*. Добавьте еще два столбца и назначьте им теги *BINi_ex_tlg_m2* и *BINi_ex_tlg_m3*, введите их имена и выберите цвет. 

Шаг	Действие: Конфигурирования окна таблицы
5	Индивидуальная настройка столбцов. Для этого используется расширенное диалоговое окно, которое открывается DD (двойным щелчком мыши) на элементе управления. Окно свойств, описанное выше, можно открыть с помощью DD (двойного щелчка мыши) и клавиши CTRL. Расширенное окно свойств в дополнение к уже описанным закладкам *General Information (Общая информация)* и *Columns (Столбцы)* содержит еще три. На закладке *Column (Столбец)* производятся следующие изменения: *Format (Формат)* объекта *Time Display (Индикатор времени)* установлен в *hh:mm:ss*. В поле *Orientation (Ориентация)* объектов *Time Display (Индикатор времени)* и *Value Display (Индикатор значений)* выбран вариант *Centered (По центру)*. *Range (Период)* установлен в *1 X 1 Час*. Для остальных опций оставлены значения по умолчанию. Окно свойств закрывается нажатием на *OK*. 

Замечание:

Краткое описание реализации подобной конфигурации в *графическом дизайнере* для кадра *ex_3_chapter_01e.PDL* можно найти в главе Bar Display (Гистограммы) (ex_3_chapter_01e).

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

Подобная реализация допускает запуск архивации в определенное время. Кроме того, значения могут сохраняться в архиве каждую минуту, каждый час и т.д.

Измените длину цикла архивации, учитывая начальное время и используемое базовое время.

4.1.7 Экспорт архивов (ex_3_chapter_01f.pdl)

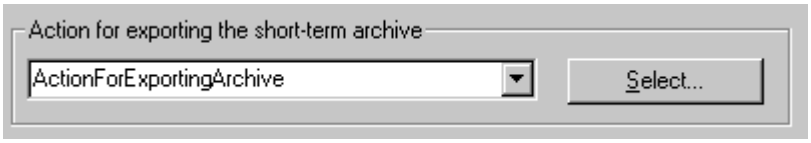
Постановка задачи

Непрерывный циклический архив значений процесса при достижении максимального числа записей нужно экспортировать в CSV–файл. Архив должен блокироваться при запуске системы и разблокироваться только по нажатию соответствующей кнопки.
Архивируемые значения должны отображаться в виде таблицы, должны быть созданы определенные пользователем панель инструментов и строка статуса. О времени экспорта пользователя должно уведомлять информационное окно.

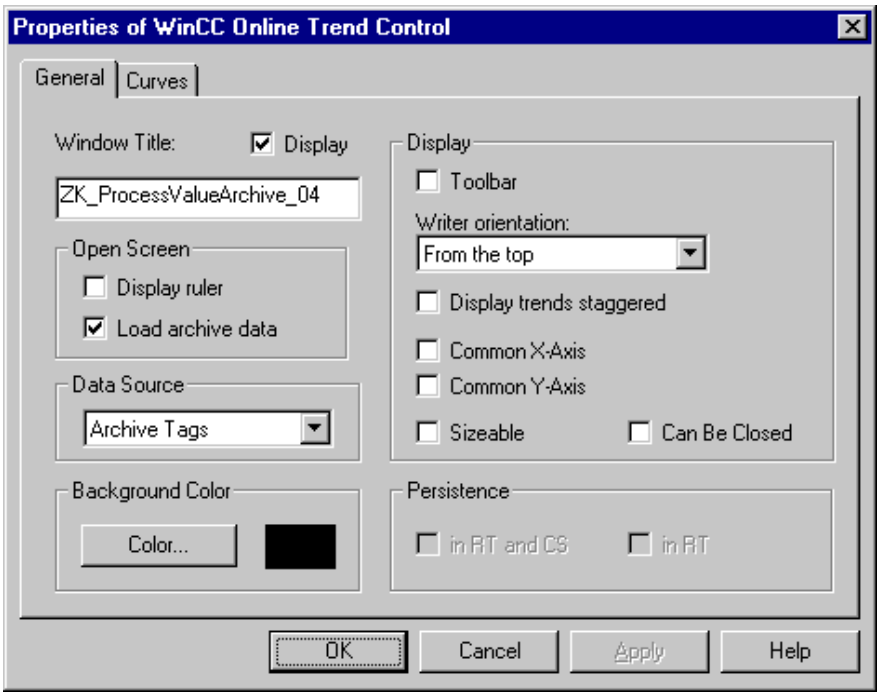
Концепция реализации

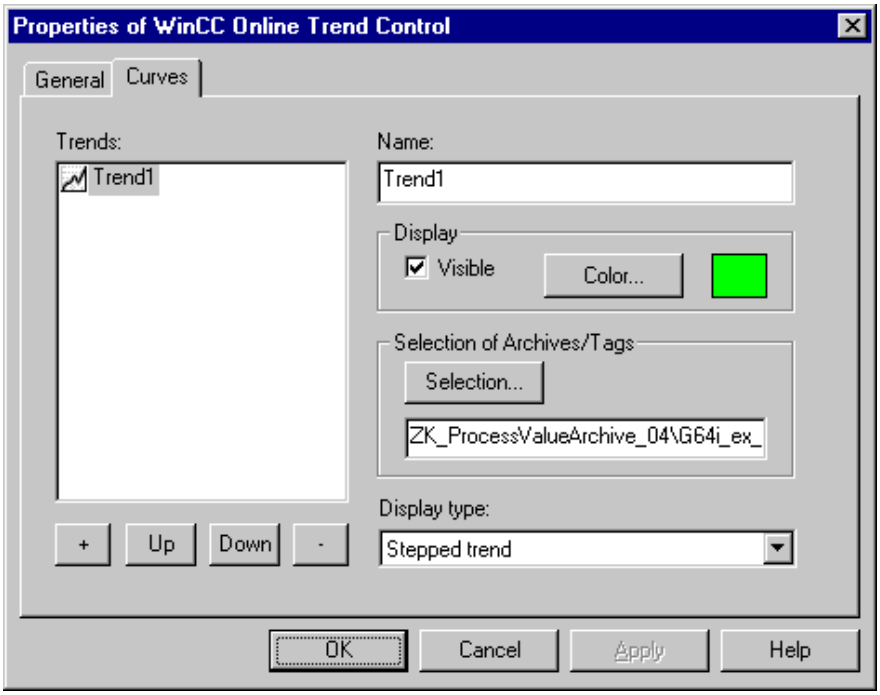
Для архивирования данных в редакторе *Tag Logging* создается непрерывный циклический архив значений процесса. Для экспортирования и его блокировки/разблокировки используются функции проекта.
Для отображения данных используется объект *WinCC Online Trend Control*.
Панель инструментов состоит из объектов *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)* и *Smart Objects (Интеллектуальные объекты)* → *Status Displays (Индикаторы состояния)*.

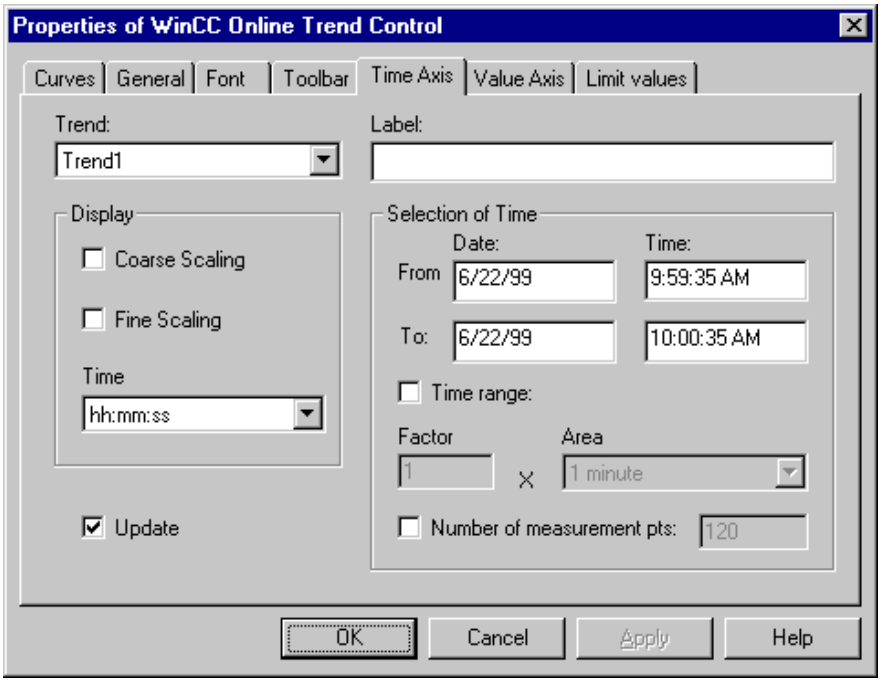
Создание архива значений процесса

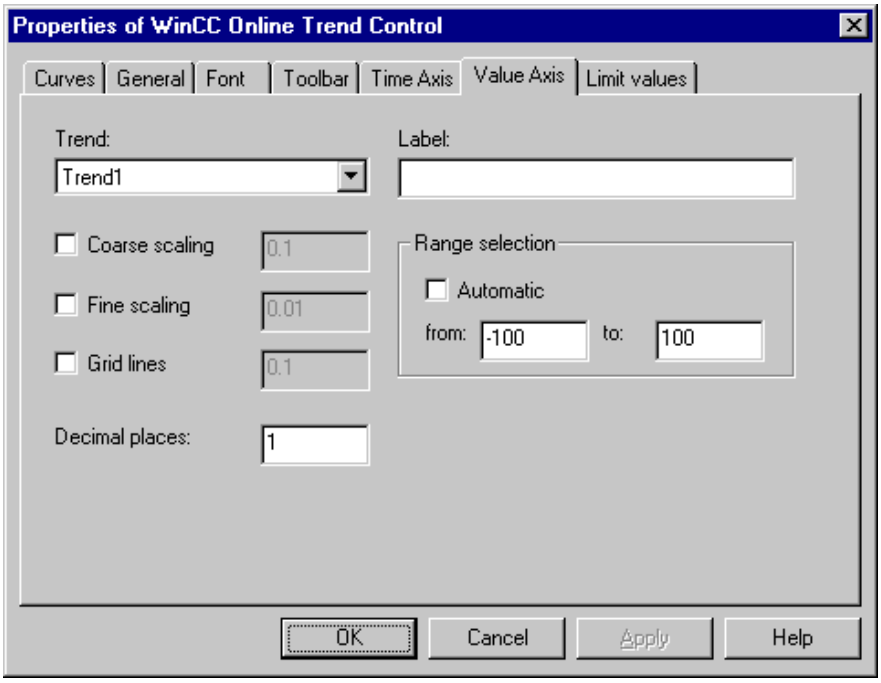
Шаг	Действие: Создание архива значений процесса
1	С помощью мастера архивов создайте новый *архив значений процесса*. В данном примере это архив *ZK_ProcessValueArchive_04*. В качестве архивируемого тега выберите *G64i_ex_tlg_04*. В данном примере его значение — это сумма трех переменных, формируемых имитатором.
2	В окне свойств архива укажите его длину — *200* записей. В качестве процедуры *Action for Exporting the Short–Term Archive (Процедура для экспорта краткосрочного архив)* используется функция *ActionForExportingArchive*. Для остальных опций оставьте значения по умолчанию. 
3	В окне свойств тега процесса оставьте значения по умолчанию.

Конфигурирование окна трендов

Шаг	Действие: Конфигурирование окна трендов
1	Создайте новый кадр; в данном примере это кадр *ex_3_chapter_01f.pdl*.
2	Конфигурирование объекта *WinCC Online Trend Control*, используемого для отображения. Он выбирается из *Object Palette (Палитры объектов)* и помещается в кадр.
3	После добавления объекта в кадр автоматически открывается конфигурационное окно. На закладке *General Information (Общая информация)* указывается имя окна, в данном примере — *ZK_ProcessValueArchive_04*. В качестве *Background Color (Цвета фона)* выбран черный. В разделе *Display (Отображение)* отключены все опции, ориентация текста – *from the top (сверху вниз)*. 

Шаг	Действие: Конфигурирование окна трендов
4	На закладке *Trends (Тренды)* производится настройка трендов. В данном примере достаточно только одного тренда. Один тренд уже создан; он переименован в *Trend1*. С помощью кнопки *Selection (Выбор)* тренду назначается *архивный тег*. В данном примере это тег *Archive Tag G64_ex_tlg_04* уже созданного архива *ZK_ProcessValueArchive_04*. В поле *Display Type (Тип отображения)* выбирается вариант *Stepped Trend (Ступенчатый тренд)*. 

Шаг	Действие: Конфигурирование окна трендов
5	Индивидуальная настройка столбцов. Для этого используется расширенное диалоговое окно, которое открывается D (двойным щелчком мыши) на элементе управления. Окно свойств, описанное выше, можно открыть с помощью D (двойного щелчка мыши) и клавиши CTRL. Расширенное окно свойств в дополнение к уже описанным закладкам *General Information (Общая информация)* и *Trends (Тренды)* содержит еще пять. На закладке *Time Axis (Ось времени)* производятся следующие изменения: опции *Coarse Grid (Крупная сетка)* и *Fine Grid (Мелкая сетка)* отключаются, *Time Format (Формат времени)* устанавливается в *hh:mm:ss*. В поле *Time Selection (Выбор времени)* отключается опция *Time Range (Период)*. 

Шаг	Действие: Конфигурирование окна трендов
6	На закладке *Value Axis (Ось значений)* в области *Trend (Тренд)* отключаются все опции. В поле *Range Selection (Выбор диапазона)* отключается опция *Automatic (Автоматический)* и вводится промежуток от *–100 до 100*. Для остальных опций оставлены значения по умолчанию. Окно свойств закрывается нажатием на *OK*. 

Функция проекта для экспорта архива

```
void ActionForExportingArchive (LPTSTR lpszArchivNameReturn, LPTSTR lpszVar
{
BOOL fRet;
int  iTlgCon = 0;
CMN_ERROR Error;
char szProj[MAX_PATH];
char szFile[MAX_PATH];
LPTSTR  lpszArchivName =
      "PDE#HD#ZK_ProcessValueArchive_04#G64i_ex_tlg_04" ;
char    szFileName[MAX_PATH] = "";
LPTSTR  lpszFileName;
TLG_IO_BACKUP_SELECT   ibs;
DWORD dwSize;
time_t          Time;
struct tm*      TimeStruct;
int nPathLen,nFileLen;

DMGetRuntimeProject( szProj, MAX_PATH, &Error);

nPathLen=strlen(szProj);
nFileLen=strlen((strrchr(szProj,\\)+1));

strncat(szFile,szProj,nPathLen-nFileLen);
sprintf(szFileName,"%s%s",szFile,"ArchiveBackUp.CSV");
lpszFileName=&szFileName[0];

time(&Time);
TimeStruct = localtime(&Time);

ibs.sysFrom.wYear   = 1997;
ibs.sysFrom.wMonth  = 1;
ibs.sysFrom.wDay    = 1;
ibs.sysFrom.wHour   = 0;
ibs.sysFrom.wMinute = 0;
ibs.sysFrom.wSecond = 0;

ibs.sysTo.wYear   = (WORD)(TimeStruct->tm_year+1900);
ibs.sysTo.wMonth  = (WORD)(TimeStruct->tm_mon+1);
ibs.sysTo.wDay    = (WORD)(TimeStruct->tm_mday);
ibs.sysTo.wHour   = (WORD)(TimeStruct->tm_hour);
ibs.sysTo.wMinute = (WORD)(TimeStruct->tm_min);
ibs.sysTo.wSecond = (WORD)(TimeStruct->tm_sec);

fRet = TLGConnect( NULL, &Error );
if (fRet==FALSE)       printf("Error in TLGConect(...)\r\n");

fRet=TLGGetBackupSize (lpszArchivName, &dwSize, &ibs, TLG_BACKUP_EVACUATE,
if (fRet==FALSE) printf("Error in TLGGetBackupSize(...)[%s]\r\n",Error.szEr
else SetTagWord("U16i_ex_tlg_00",(WORD)dwSize);

fRet=TLGBackup (lpszArchivName, lpszFileName, &ibs, TLG_BACKUP_EXPORT, TLG_
if (fRet==FALSE) printf("Error in TLGBackup(...) [%s]\r\n",Error.szErrorTex
else SetTagBit("BINi_ex_tlg_09",TRUE);

TLGDisconnect( NULL );
}
```

Определение пути проекта.

Создание файла с определенным именем для хранения архива. Имя также включает в себя путь.

Определение системного времени.

Указание начальной и конечной временных меток, между которыми находятся архивируемые значения.

Установка соединения с подсистемой *Tag Logging* с помощью функции *TLGConnect*.

Определение размера экспортируемых данных с помощью функции *TLGBackupSize*. Полученное значение сохраняется во внутреннем теге.

Экспорт архива с помощью функции *TLGBackup* и установка в 1 двоичного тега *BINi_ex_tlg_09*.

Реализация в графическом дизайнере

Шаг	Действие: Реализация в графическом дизайнере
1	В м*енеджере тегов* создайте три двоичных тега. В данном примере это теги *BINi_ex_tlg_06*, *BINi_ex_tlg_08* и *BINi_ex_tlg_09*. Также требуется еще один тег U16i_ex_tlg_00 типа *Unsigned 16–Bit Value (16–битная величина без знака)*.
2	Реализация панели инструментов была подробно описана в примере в главе Архивация при превышении значения (ex_3_chapter_01b.pdl). В данной главе описываются только новые элементы управления.
3	Для управления архивированием используется объект *Windows Object (Объект Windows)* → *Button (Кнопка)*. В данном примере это *Button16*. Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создается *процедура Си*, которая инвертирует тег *BINi_ex_tlg_08* и вызывает функцию проекта *LockUnlockArchive*. Двоичный тег используется для хранения текущего состояния архива. 
4	Создайте новый кадр, который будет отображаться в момент экспорта архива. В данном примере для этого используется кадр *ex_5_window_03.PDL*. Этот кадр содержит объект *Standard Object (Стандартный объект)* → *Static Text (Статический текст)*, который с помощью процедуры Си будет отображать сообщение. Текст состоит из фиксированной части и числового значения тега *U16i_ex_tlg_00*. Этот тег содержит размер экспортируемых данных. Также кадр содержит объекты *Windows Object (Объект Windows)* → *Button (Кнопка)* и *Smart Object (Интеллектуальный объект)* → *Graphic Object (Графический объект)*, которые через *прямое соединение* для события *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* сбрасывают тег *BINi_ex_tlg_09* в 0. 
5	В начальном кадре сконфигурируйте *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*; в данном примере это объект *Picture Window1*. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* введите имя кадра *ex_5_window_03.PDL*. В поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Display (Отображение)* создайте *соединение с тегом BINi_ex_tlg_09*.

Шаг	Действие: Реализация в графическом дизайнере
6	Для отображения строки статуса используются две *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*; в данном примере это объекты *Button14* и *Button17*. У кнопки *Button17* для *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* создайте *динамический диалог*, который, в зависимости от значения тега *BINi_ex_tlg_06*, будет возвращать либо *Update Started (Обновление запущено)*, либо *Update Stopped (Обновление остановлено)*. У *Button14* для *Properties (Свойства)* → *Font (Шрифт)* → *Text (Текст)* создайте *динамический диалог*, который, в зависимости от значения тега *BINi_ex_tlg_08*, будет возвращать либо *Archive Enabled (Архивация разрешена)*, либо *Archive Locked (Архивация заблокирована)*. 

Функция проекта для блокировки/разблокировки архива

```
void LockUnlockArchive (BOOL bLock)
{
  BOOL fRet;
  CMN_ERROR Error;
  LPTSTR       lpszArchivName = "ZK_ProcessValueArchive_04";

fRet = TLGConnect( NULL, &Error );

if (fRet==FALSE)
        printf("Error in TLGConnect(...)\r\n");
else
{
        fRet=TLGLockArchiv (NULL,lpszArchivName,bLock,&Error );
        if (fRet==FALSE) printf("Error in TLGLockArchive(...) [%s]\r\n",
        Error.szErrorText);
        TLGDisconnect( NULL );
}

}
```

Установка соединения с подсистемой *Tag Logging*.

Вызов функции *TLGLockArchive*. Переданный параметр *bLock* определяет, блокируется архив или разблокируется.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

Архивируемые теги должны быть настроены соответствующим образом.

Должны быть указан максимальный размер архива, а также имя и путь к файлу, в который архив будет экспортироваться.

4.2 Регистрация аварийных сообщений

В режиме исполнения примеры, относящиеся к данной теме, выбираются щелчком (мыши) на изображенной выше кнопке. Эти примеры сконфигурированы в серии кадров от *ex_3_chapter_02.pdl* до *ex_3_chapter_02d.pdl*..

Общая информация

Подсистема регистрации аварийных сообщений (Alarm Logging) отвечает за регистрацию и архивацию сообщений. Подсистема включает в себя функции для передачи сообщений технологического процесса, для их обработки, отображения, подтверждения и архивации.
Таким образом, система *Alarm Logging* помогает пользователю в определении причин ошибок.

4.2.1 Битовая процедура сообщения (ex_3_chapter_02.pdl)

Постановка задачи

Системой регистрации аварийных сообщений должны контролироваться четыре двигателя. Ошибки отображаются путем установки различных битов внутри тега, назначенного каждому двигателю. Статус сообщений для двигателей хранится во внутренних тегах. Изображение двигателя должно изменяться в зависимости от статуса сообщения.
Сообщения должны отображаться в окне сообщений.

Концепция реализации

В системе регистрации аварийных сообщений необходимо создать несколько сообщений, относящихся к четырем контролируемым двигателям.
Окно сообщений реализуется в графическом дизайнере с использованием элемента управления аварийными сообщениями WinCC *Alarm Control*.
Отдельные двигатели отображаются при помощи нескольких *Standard Objects (Стандартных объектов)*. Изменения изображений двигателей при различных статусах сообщений реализуются с помощью *процедур Cи*.

Создание необходимых тегов

Шаг	Процедура: Создание необходимых тегов
1	Создание двенадцати тегов типа *Unsigned 16–Bit Value (16–битная величина без знака)* в *менеджере тегов*. Четыре тега используются в качестве тегов событий. В данном примере это теги *U16i_ex_alg_00*, *U16i_ex_alg_03*, *U16i_ex_alg_06* и *U16i_ex_alg_09*. Четыре других тега используются в качестве тегов состояний, это теги *U16i_ex_alg_02*, *U16i_ex_alg_05*, *U16i_ex_alg_08* и *U16i_ex_alg_11*. Оставшиеся теги *U16i_ex_alg_12*, *U16i_ex_alg_13*, *U16i_ex_alg_14* и *U16i_ex_alg_15* являются тегами подтверждения.

Блоки сообщений

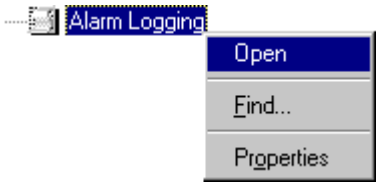
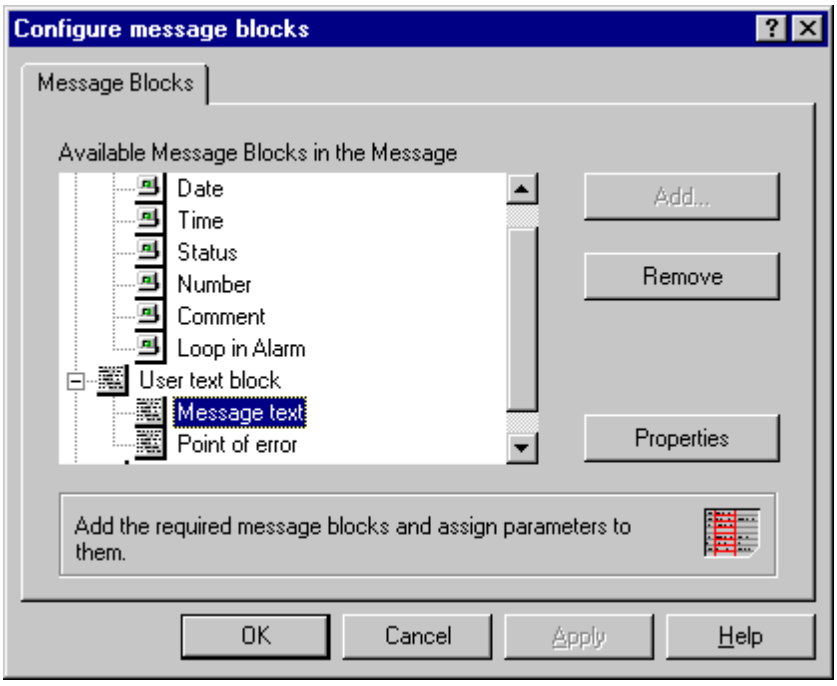
Сообщение состоит из различных блоков. Их можно разбить на три категории:

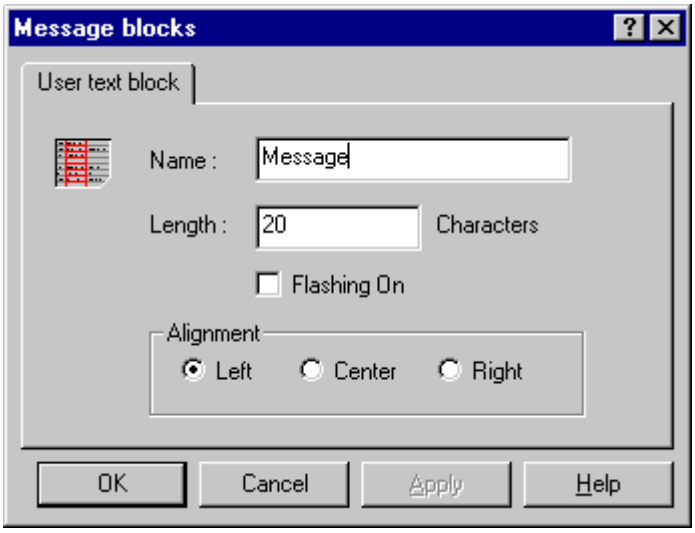
Системные блоки (System Blocks): Они содержат системные данные, которые устанавливаются системой *Alarm Logging*. Это дата, время, отчетная идентификация и т.д.

Блоки значений процесса (Process Value Blocks): Они включают значения, возвращаемые технологическим процессом, например, критические уровни заполнения, температуры, и т.д.

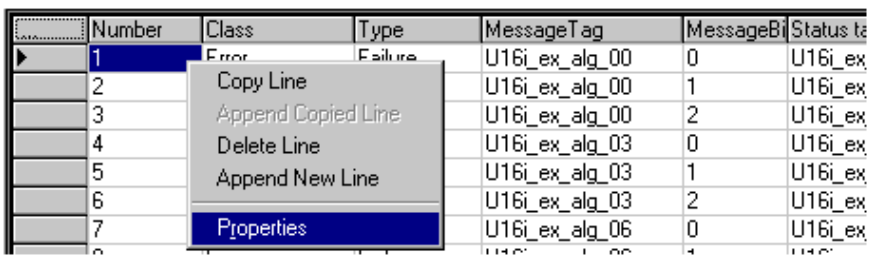
Пользовательские текстовые блоки (User Text Blocks): Тексты, представляющие собой общую и справочную информацию, например, объяснения ошибок, причины сообщений, адреса ошибок и т.д.

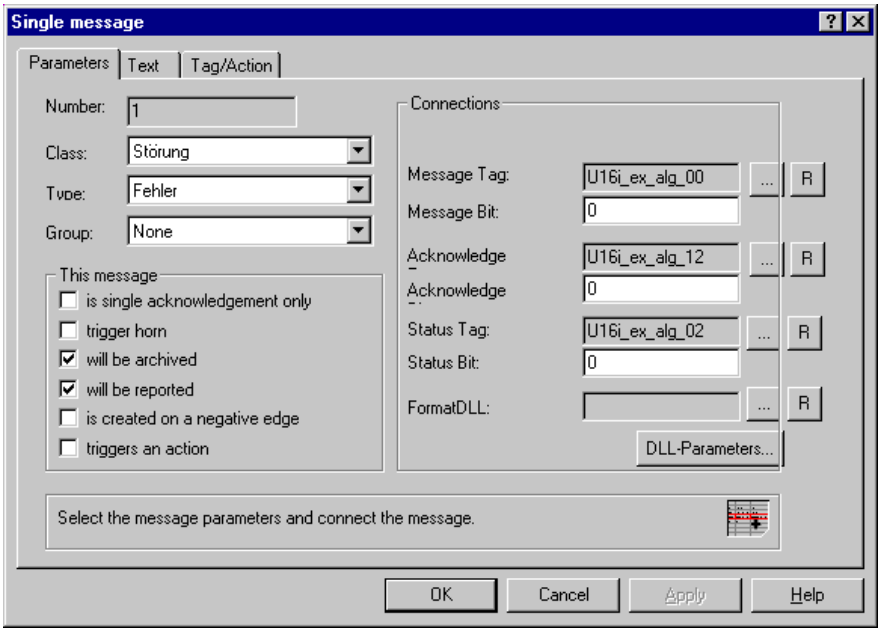
Конфигурирование блока сообщений

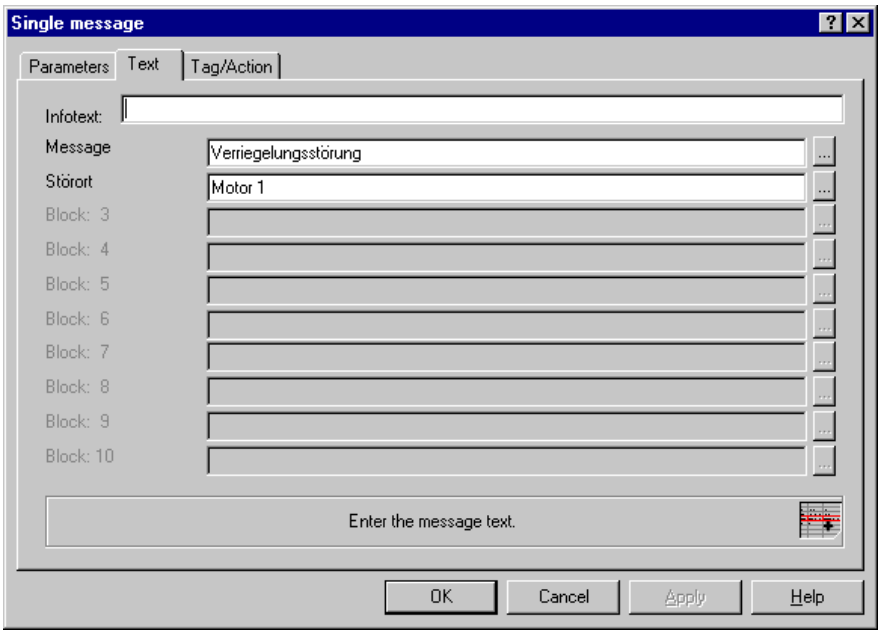
Шаг	Процедура: Конфигурирование блока сообщений
1	В *проводнике WinCC* откройте редактор системы *Alarm Logging*. Для этого щелкните R (правой кнопкой мыши) на пункте *Alarm Logging* и в появившемся меню выберите *Open (Открыть)*. 
2	Выбор требуемых блоков сообщений. Для этого щелкните R (правой кнопкой мыши) в области *Message Blocks (Блоки сообщений)* и в появившемся меню выберите *Message Blocks (Блоки сообщений)*. Откроется диалоговое окно *Configure Message Blocks (Настройка блоков сообщений)*. 
3	С помощью кнопки *Add (Добавить)* открывается диалоговое окно добавления блоков для соответствующего элемента системных блоков, пользовательских текстовых блоков или блоков значений процесса. Если блок выбран с помощью (мыши) в диалоговом окне *Configure Message Blocks (Конфигурирование блоков сообщений)*, то кнопки *Remove (Удалить)* и *Properties (Свойства)* становятся доступными. Первая кнопка позволяет удалять выбранные блоки, вторая кнопка позволяет настраивать свойства конкретных блоков сообщений. 

Шаг	Процедура: Конфигурирование блоков сообщений
4	В данном примере оставлено *Name (Имя)* оставлено без изменений, а *Length (Длина)* устанавливается равной *20* символам. Нажатие на *OK* завершает процесс настройки в диалоговом окне *Message Blocks (Блоки сообщений)*. Диалоговое окно конфигурирования блоков сообщений *Configure Message Blocks* также можно закрыть, нажав на кнопку *OK*. 

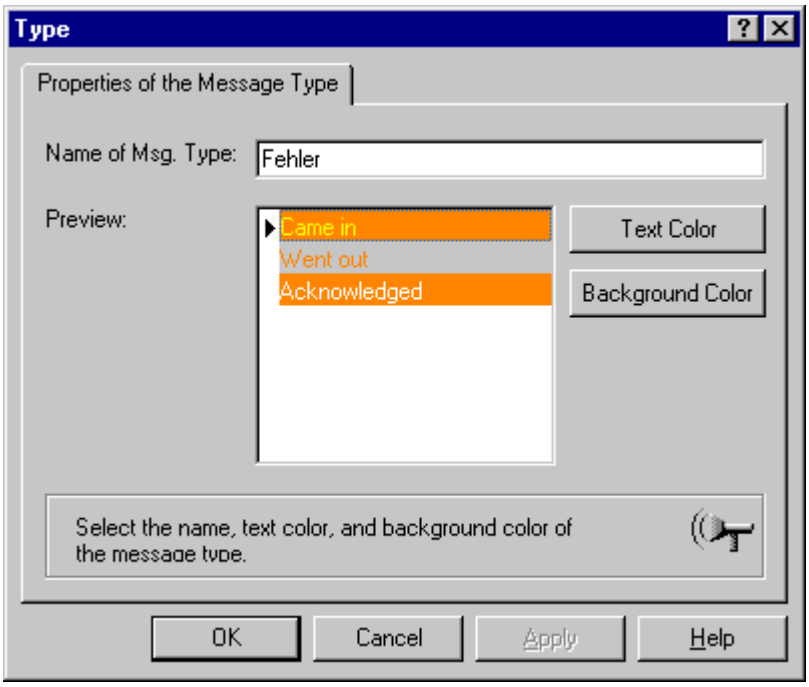
Создание одиночных сообщений

Шаг	Процедура: Создание одиночных сообщений
1	В редакторе *Alarm Logging* окно таблицы располагается в нижней части. В этой области конфигурируются одиночные сообщения и отображаются уже сконфигурированные. Новую строку можно добавить с помощью щелчка R (правой кнопки мыши).  Для этого примера создано 12 различных одиночных сообщений. Каждое сообщение соответствует одной строке в окне таблицы и состоит из ряда столбцов. Настройки могут быть сделаны непосредственно в конкретных столбцах. В этом примере, однако, настройки сделаны в диалоговом окне *Single Message (Одиночное сообщение)*. Это диалоговое окно открывается щелчком R (правой кнопки мыши) на соответствующей строке сообщения. 

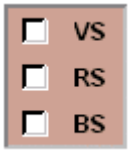
Шаг	Процедура: Конфигурирование блоков сообщений
2	Откройте диалоговое окно *Single Message (Единичное сообщение)* для первой строки, как описано в предыдущем шаге. На закладке *Parameters (Параметры)* выберите класс сообщения *Error (Ошибка)* и тип сообщения *Failure (Сбой)*. В поле *This Message (Данное сообщение)* выберите переключатели *will be archived (Будет заархивировано)* и *will be reported (Будет включено в отчет)*. В поле *Connections (Соединения)* в качестве *Event Tag (Тега события)* выберите тег *U16i_ex_alg_00*. В качестве *Event Bit (Бита события)* введите *0*. Это означает, что сообщение будет сгенерировано в том случае, если первый бит тега будет установлен в 1. В качестве *Acknowledge Tag (Тега подтверждения)* выбирается тег *U16i_ex_alg_12* и в качестве *Acknowledge Bit (Бита подтверждения)* вводится *0*. Другими словами, если сообщение подтверждается в режиме исполнения, то первый бит тега устанавливается в 1. В качестве *Status Tag (Тега состояния)* выбирается тег *U16i_ex_alg_02* и в качестве *Status Bit (Бита состояния)* вводится *0*. Это означает, что первый бит тега представляет собой состояние сообщения *Came in/Went out Status (Пришло / Ушло)*. Если сообщение находится в состоянии ожидания, этот бит устанавливается в 1, и если сообщение выходит из ожидания, этот бит сбрасывается. Девятый бит тега содержит *Acknowledge Status (Состояние подтверждения)* сообщения. Если оно не подтверждается, бит имеет состояние 1, если подтверждается, состояние 0. Тег состояния из 16 битов может содержать состояние 8 одиночных сообщений. Младший байт содержит состояние *Came in/Went out Status (Пришло / Ушло)*, а старший байт — *Acknowledge Status (Состояние подтверждения)*. 

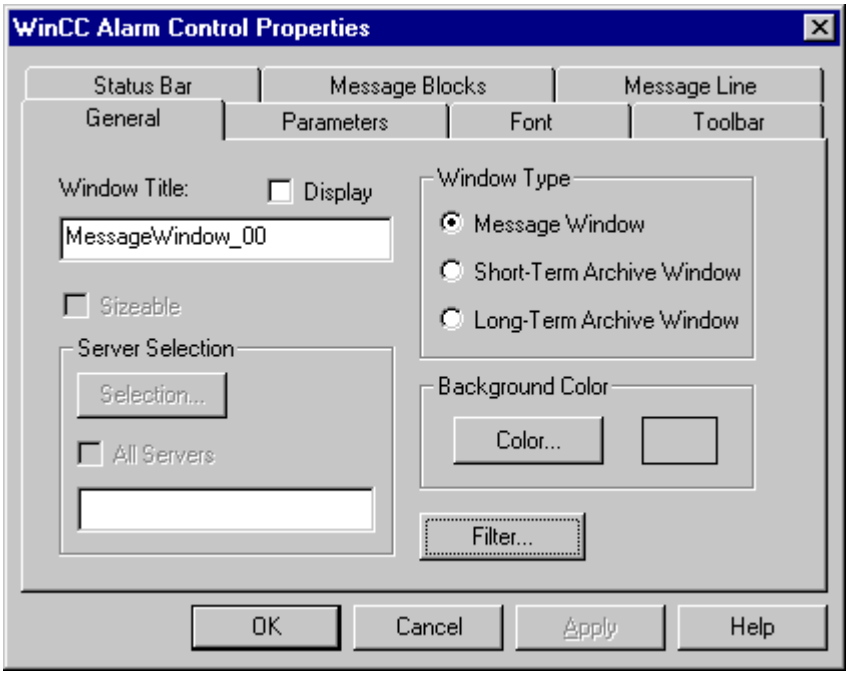
Шаг	Процедура: Конфигурирование блоков сообщений
3	На закладке *Text (Текст)* в строке *Message Text (Текст сообщения)* вводится *Locking Error (Блокирующая ошибка)* и в строке *Point of Error (Место ошибки)* вводится *Motor 1 (Двигатель 1)*. Поле *Infotext (Комментарий)* не используется. Закладка *Tag/Action (Тег/Процедура)* для этого примера заполняться не должна. Для применения сделанных настроек необходимо нажать на кнопку *OK*. 
4	Только что созданное сообщение позволяет контролировать только первый из четырех двигателей. Для него же создаются еще две строки сообщений. Настройки делаются аналогично тому, как это описано в шагах 2 и 3, но *Event Bits (Биты событий)*, *Acknowledge Bits (Биты подтверждений)* и *Status Bits (Биты состояний)* настраиваются по–другому. Также используются *Feedback Error (Ошибка обратной связи)* и *Bimetal Error (Биметаллическая ошибка)*.
5	Для трех других двигателей также создаются по три строки сообщений на каждый двигатель. Для каждого двигателя нужно также настроить *Status Tags (Теги состояний)* *Event Tags (Теги событий)*, *Acknowledge Tags (Теги подтверждений)* и тексты для *Point of Error (Места ошибки)*. ... | Number | Class | Type | MessageTag | MessageBi | Status tag | Status bit | Message text 1 | Error | Failure | U16i_ex_alg_00 | 0 | U16i_ex_alg_02 | 0 | Lock Error 2 | Error | Failure | U16i_ex_alg_00 | 1 | U16i_ex_alg_02 | 1 | Feedback Error 3 | Error | Failure | U16i_ex_alg_00 | 2 | U16i_ex_alg_02 | 2 | Bimetal Error 4 | Error | Failure | U16i_ex_alg_03 | 0 | U16i_ex_alg_05 | 0 | Lock Error 5 | Error | Failure | U16i_ex_alg_03 | 1 | U16i_ex_alg_05 | 1 | Feedback Error 6 | Error | Failure | U16i_ex_alg_03 | 2 | U16i_ex_alg_05 | 2 | Bimetal Error 7 | Error | Failure | U16i_ex_alg_06 | 0 | U16i_ex_alg_08 | 0 | Lock Error 8 | Error | Failure | U16i_ex_alg_06 | 1 | U16i_ex_alg_08 | 1 | Feedback Error 9 | Error | Failure | U16i_ex_alg_06 | 2 | U16i_ex_alg_08 | 2 | Bimetal Error 10 | Error | Failure | U16i_ex_alg_09 | 0 | U16i_ex_alg_11 | 0 | Lock Error 11 | Error | Failure | U16i_ex_alg_09 | 1 | U16i_ex_alg_11 | 1 | Feedback Error 12 | Error | Failure | U16i_ex_alg_09 | 2 | U16i_ex_alg_11 | 2 | Bimetal Error

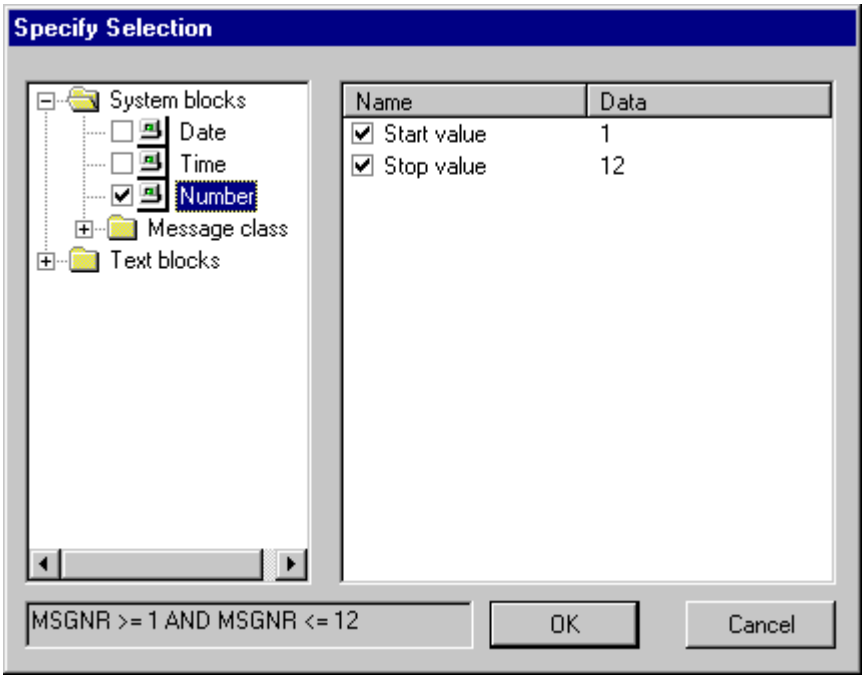
Конфигурирование цветовой схемы сообщений

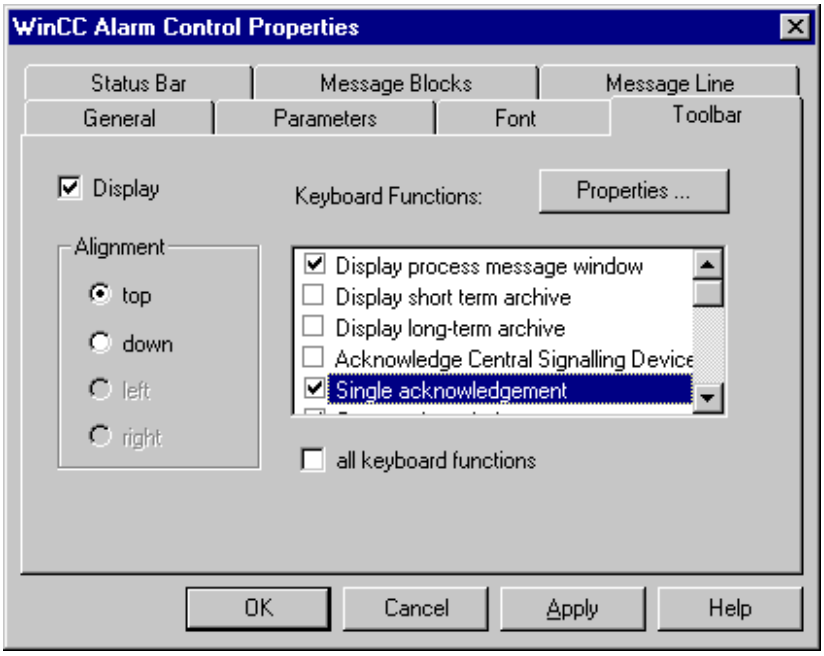
Шаг	Процедура: Конфигурирование цветовой схемы сообщений
1	Сконфигурированные одиночные сообщения имеют класс сообщения *Error (Ошибка)* и тип сообщения *Failure (Сбой)*. По щелчку (мыши) на элементе *Message Classes (Классы сообщений)*, все доступные классы сообщений будут отображены в правом окне. После D (двойного щелчка мыши) на иконке класса сообщений *Error (Ошибка)* будут отображены все типы сообщений, имеющие отношение к этому классу. Диалоговое окно *Type (Тип)* открывается посредством D (двойного щелчка мыши) на иконке типа сообщения *Failure (Сбой)* или щелчком на ней R (правой кнопки мыши) с выбором соответствующего пункта всплывающего меню. 
2	В диалоговом окне *Type (Тип)* для каждого статуса сообщения может быть создана цветовая схема. Например, используется следующая цветовая схема: Came in: Текст = Желтый, Фон = Оранжевый Went out: Текст = Оранжевый, Фон = Светло–серый Acknowledged: Текст = Белый, Фон = Оранжевый 
3	Настройки, сделанные в *Alarm Logging*, сохраняются при помощи команды меню *File (Файл)* → *Save (Сохранить)*.

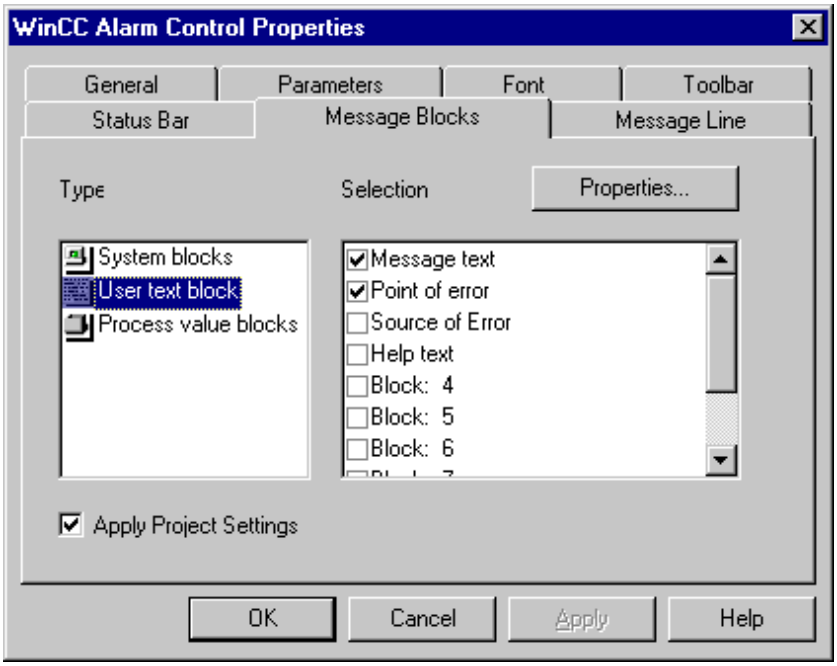
Реализация в графическом дизайнере

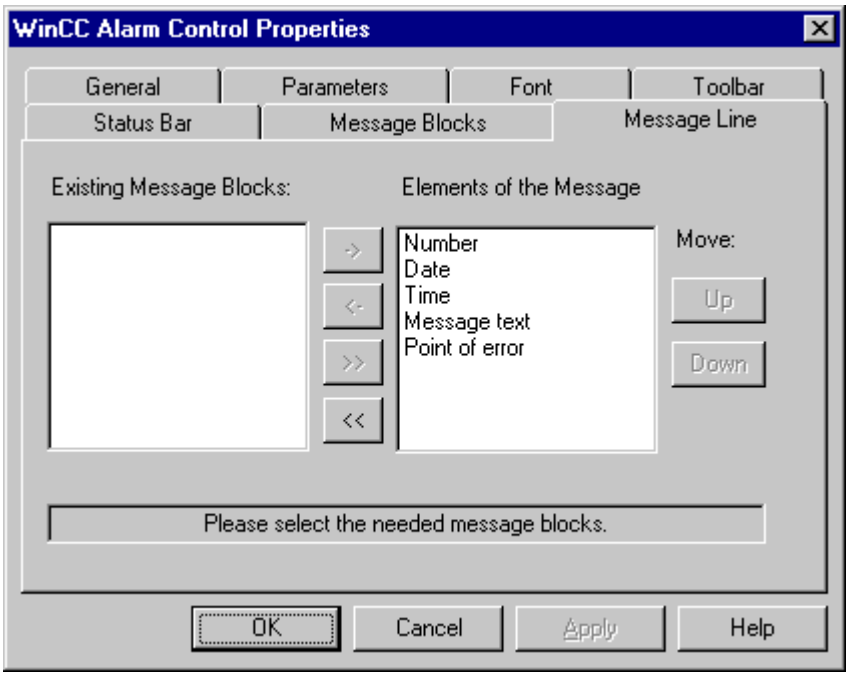
Шаг	Процедура: Реализация в графическом дизайнере
1	Создание нового кадра, в нашем проекте это кадр *ex_3_chapter_02*.
2	Отдельные двигатели отображаются при помощи объектов *Standard Object (Стандартный объект)* → *Circle (Окружность)*, *Standard Object (Стандартный объект)* → *Static Text (Статический текст)* и *Standard Object (Стандартный объект)* → *Polygon (Многоугольник)*. При возникновении ошибки или при подтверждении сообщения двигатель меняет свою цветовую схему. Эта цветовая схема должна соответствовать состояниям сообщений came in (пришло), went out (ушло) и acknowledged (подтверждено). Для этого у *статического текста* в *Properties (Свойства)* → *Colors (Цвета)* → *Font Color (Цвет шрифта)* создается *процедура Cu*, которая изменяет цвет шрифта в зависимости от текущего состояния тега статуса, соответствующего двигателю. Аналогично у *Circle (Окружности)* в *Properties (Свойства)* → *Colors (Цвета)* → *Background Color (Цвет фона)* создается *процедура Cu*, которая выполняет ту же самую задачу. 
3	Возникновение ошибки в двигателе моделируется с помощью *Windows Object (Объект Windows)* → *Check–Box (Группа флажков)*. В поле *Properties (Свойства)* → *Geometry (Геометрия)* → *Number of Boxes (Количество флажков)* вводится *3*. Для атрибута *Properties (Свойства)* → *Output/Input (Вывод/Ввод)* → *Selected Boxes (Выбранные флажки)* создается *соединение с тегом* соответствующего события двигателя. 

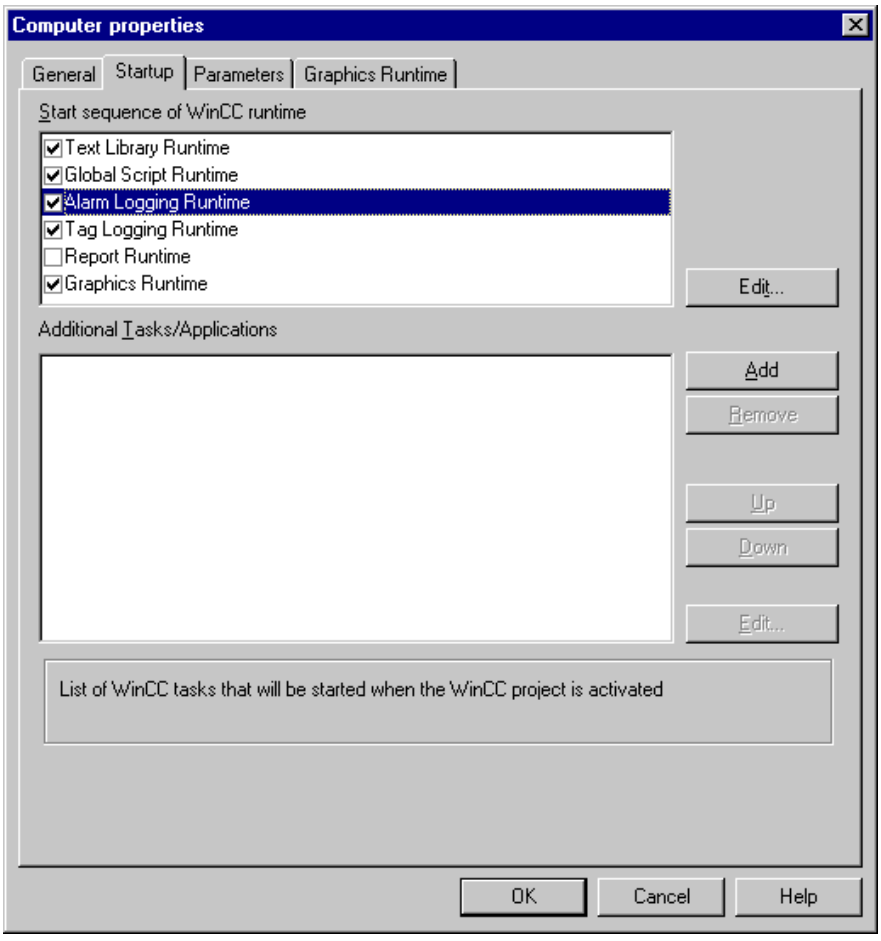
Шаг	Процедура: Реализация в графическом дизайнере
4	Для отображения сообщений, сконфигурированных в *Alarm Logging*, используется объект *WinCC Alarm Control*. Он выбирается из палитры объектов закладки *Controls (Элементы управления)* и затем помещается в кадр. 
5	После размещения элемента управления в кадре, его *конфигурационное диалоговое окно* будет отображено автоматически. Это диалоговое окно может быть закрыто нажатием на *OK*. Откройте диалоговое окно свойств элемента управления. Это окно отображается после D (двойного щелчка мыши) на данном элементе управления. На закладке *General Information (Общая информация)* кнопка *Selection (Выбор)* используется для выбора одиночных сообщений, созданных в *Alarm Logging*. 

Шаг	Процедура: Реализация в графическом дизайнере
6	После нажатия (мыши) на системном блоке *Number (Номер)* в правом окне будут отображены 2 переключателя. *Start Value (Начальное значение)* изменяется на *1* посредством D (двойного щелчка мыши) на значении по умолчанию и *Stop Value (Конечное значение)* меняется на *12*. Это означает, что данный элемент управления отображает только одиночные сообщения с номерами от 1 до 12. 

Шаг	Процедура: Реализация в графическом дизайнере
7	На закладке *Toolbar (Панель инструментов)* выбираются кнопки, которые будут отображаться в режиме исполнения. В данном примере необходимы следующие кнопки: *Single Acknowledgment (Одиночное подтверждение)*, *Group Acknowledgment (Групповое подтверждение)*, *Auto Scroll On/Off (Авто скроллинг Вкл/Выкл)*, *Beginning of the List (Начало списка)*, *End of the List (Конец списка)*, *Next Message (Следующее сообщение)* и *Previous Message (Предыдущее сообщение)*. 

Шаг	Процедура: Реализация в графическом дизайнере
8	На закладке *Message Blocks (Блоки сообщений)* выбираются столбцы, которые будут отображаться в строке сообщений. В данном примере системные блоки выбираются (мышью) в поле *Type (Тип)*. В правом окне выбраны *Date (Дата)*, *Time (Время)* и *Number (Номер)*. Для пользовательских текстовых блоков (User Text Blocks) выбираются *Message Text (Текст сообщения)* и *Point of Error (Место ошибки)*. 

Шаг	Процедура: Реализация в графическом дизайнере
9	На закладке *Message Line (Строка сообщения)* предварительно выбранные блоки сообщений добавляются к строке сообщений. Поле *Available Message Blocks (Доступные блоки сообщений)* содержит список доступных значений. По нажатию на кнопку –> каждый конкретный блок сообщения может быть добавлен в строку сообщений. Нажатием на кнопку >> в строку сообщений добавляются все перечисленные в окне блоки сообщений. Выход из диалогового окна свойств производится по кнопке *OK*. 

Шаг	Процедура: Реализация в графическом дизайнере
10	Активизация *Alarm Logging Runtime (Режима исполнения системы регистрации аварийных сообщений)*. Для этого нажмите R (правой кнопкой мыши) на элементе *Computer (Компьютер)* в *проводнике WinCC* и во всплывающем меню выберите пункт *Properties (Свойства)*, после чего будет открыто диалоговое окно *Computer List Properties (Свойства списка компьютеров)*. По нажатию на кнопку *Properties (Свойства)* открывается диалоговое окно свойств локального компьютера. На закладке *Startup (Запуск)* выбираются приложения, которые будут активизированы в режиме исполнения. Переключатель *Alarm Logging Runtime (Режим исполнения системы регистрации аварийных сообщений)* должен быть выбран. Диалоговые окна *Computer Properties (Свойства компьютера)* и *Computer List Properties (Свойства списка компьютеров)* можно закрыть, нажав на кнопку *OK*. 

Процедура Си для Circle1

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
DWORD state;

state=GetTagDWord ("U16i_ex_alg_02");

if ((state&1)||(state&2)||(state&4))
        return 0x80FF;
else
        return 0xFFFFFF;

}
```

Эта *процедура Си* делает динамическим свойство “Цвет фона” для круга, назначенного первому двигателю.

Считывается тег состояния *U16i_ex_alg_02*, назначенный первому двигателю. Младший байт этого тега содержит статус сообщения *came in/went out (пришло/ушло)*, т. е. если первый, второй или третий бит данного тега установлены в 1, то сообщение находится с состоянии ожидания и цвет фона круга будет оранжевым. (hex 80ff). Если сообщение выйдет из состояния ожидания, то будет установлен белый цвет фона (hex ffffff).

Данная *процедура Си* запускается по изменению тега состояния *U16i_ex_alg_02*.

Процедура Си для StaticText1

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
DWORD state;

State=GetTagDWord ("U16i_ex_alg_02");

if ( ((state&1)&&(state&256)) || ((state&2)&&(state&512)) ||
        ((state&4)&&(state&1024)) )
        return 0xFFFF;
else if ((state&1)||(state&2)||(state&4))
        return 0xFFFFFF;
else if ((state&256)||(state&512)||(state&1024))
        return 0x80FF;
else    return 0x800000;

}
```

Данная *процедура Си* делает динамическим свойство “Цвет шрифта” для статического текста, назначенного первому двигателю.

Считывается назначенный первому двигателю тег состояния *U16i_ex_alg_02*. Младший байт этого тега содержит значение состояния *came in/went out (пришло/ушло)* для сообщения, старший байт — значение состояния *acknowledged (подтверждено)* для сообщения. Если неподтвержденное сообщение находится в состоянии ожидания, цвет шрифта устанавливается в желтый (hex ffff); в случае подтверждения сообщения цвет шрифта устанавливается в белый (hex ffffff); в случае неподтвержденного сообщения, но в состоянии *went out (ушло)* цвет шрифта устанавливается в оранжевый (hex 80ff). В нормальном случае цвет шрифта темно–синий (hex 800000).

Данная *процедура Си* запускается по изменению тега состояния *U16i_ex_alg_02*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:
В общем случае перед использованием описанных приемов необходимо учесть следующее:

Требуемые блоки сообщений должны быть настроены в соответствии с вашими требованиями.

Теги события, состояния и подтверждения, также как и их биты, должны быть настроены в соответствии с вашими требованиями.

4.2.2 Контроль по уставкам (ex_3_chapter_02a.pdl)

Постановка задачи

Система *Alarm Logging (Регистрации аварийных сообщений)* должна проверять значения давления и температуры в трех контейнерах. Если проверяемые аналоговые значения находятся близко к критическому диапазону, должны генерироваться предупреждения. Если они достигают критического диапазона, должны генерироваться аварийные сообщения. Возникновение аварийного сообщения должно сопровождаться световой и звуковой сигнализацией в графическом дизайнере.
Используется макет окна сообщений в значительной мере определяемый пользователем.

Концепция реализации

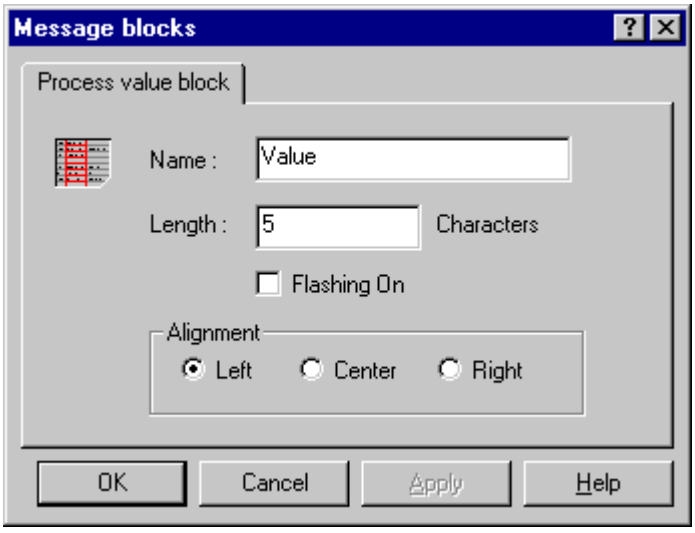
В *Alarm Logging* необходимо создать несколько сообщений, которые относятся к трем проверяемым контейнерам.
Окно сообщений создается в графическом дизайнере с использованием объекта *WinCC Alarm Control*. Панель инструментов состоит из нескольких объектов *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)* и *Smart Objects (Интеллектуальные объекты)* → *Status Displays (Индикаторы состояния)*.

Создание необходимых тегов

Шаг	Процедура: Создание необходимых тегов
1	Создание шести тегов типа *Unsigned 16–Bit Value (16–битовая величина без знака)* в *менеджере тегов*. Три из них содержат значения температур контейнеров. В примере это теги *U16i_ex_alg_t1*, *U16i_ex_alg_t2* и *U16i_ex_alg_t3*. Каждый из трех оставшихся тегов содержит значения давлений. В примере это теги *U16i_ex_alg_p1*, *U16i_ex_alg_p2* и *U16i_ex_alg_p3*. Требуются три дополнительных тега типа *Unsigned 16–Bit Value (16–битовая величина без знака)*, которые используются как теги состояний. В данном случае это теги *U16i_ex_alg_01*, *U16i_ex_alg_04* и *U16i_ex_alg_07*. Один тег типа *Unsigned 16–Bit Value (16–битовая величина без знака)* необходим для контроля центрального индикатора; в примере это тег *U16i_ex_alg_10*. Дополнительно требуется два тега типа *Binary Tag (Двоичный тег)*. В данном случае это теги *BINi_ex_alg_00* и *BINi_ex_alg_03*.

Замечание:
Настройки, сделанные в таблице *Configure Message Blocks (Конфигурирование блоков сообщений)* предыдущего примера, считаются завершенными, и в дальнейшем повторно описываться не будут.

Создание нового шаблона окна сообщения

Шаг	Процедура: Создание нового шаблона окна сообщения
1	Откройте редактор *Alarm Logging*. Если приходит сообщение, то окно сообщений отображает текущее значение проверяемого тега. Для этого должен быть создан новый блок значений процесса. С помощью щелчка R (правой кнопки мыши) на элементе *Message Blocks (Блоки сообщений)* откройте диалоговое окно *Configure Message Blocks (Конфигурирование блоков сообщений)*. Выберите закладку *Process Value Blocks (Блоки значений процесса)* и, нажав кнопку *Add (Добавить)*, откройте диалоговое окно *Add Process Value Blocks... (Добавление блоков значений процесса...)*. Добавляется новый блок значений процесса. Диалоговое окно закрывается нажатием на *OK*. После D (двойного щелчка мыши) на элементе *Process Value Blocks (Блоки значений процесса)* будет отображен новый блок. Если этот блок выбран, диалоговое окно его свойств можно вызвать с помощью кнопки *Properties (Свойства)*. В данном примере в поле *Name (Имя)* было введено *Value*, а в поле *Length (Длина)* — 5 символов. Нажав на кнопку *OK* можно применить установки, сделанные в диалоговых окнах *Message Blocks (Блоки сообщений)* и *Configure Message Blocks (Конфигурирование блоков сообщений)*. 

Общая информация

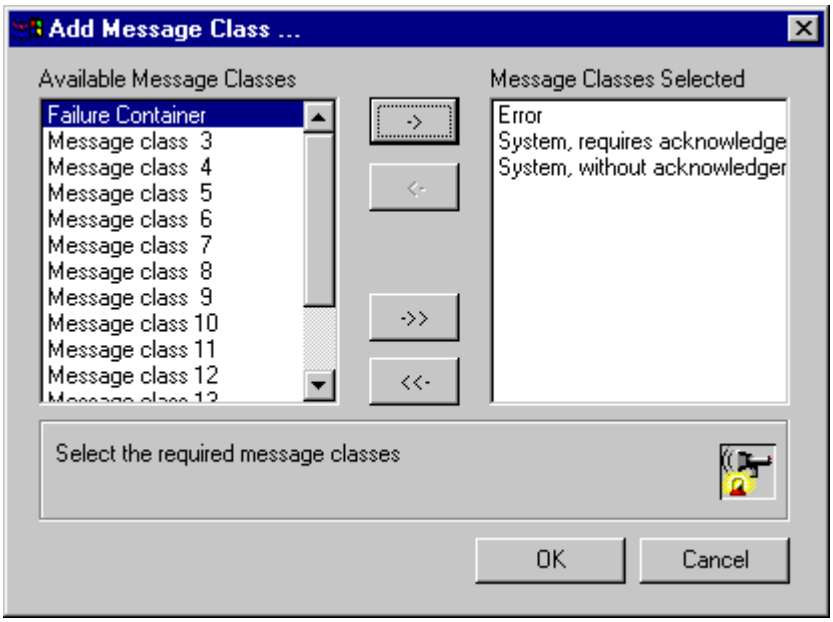
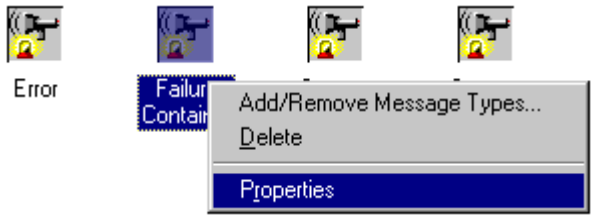
С помощью классов сообщений для всех типов сообщений, принадлежащих данному классу, можно указать:

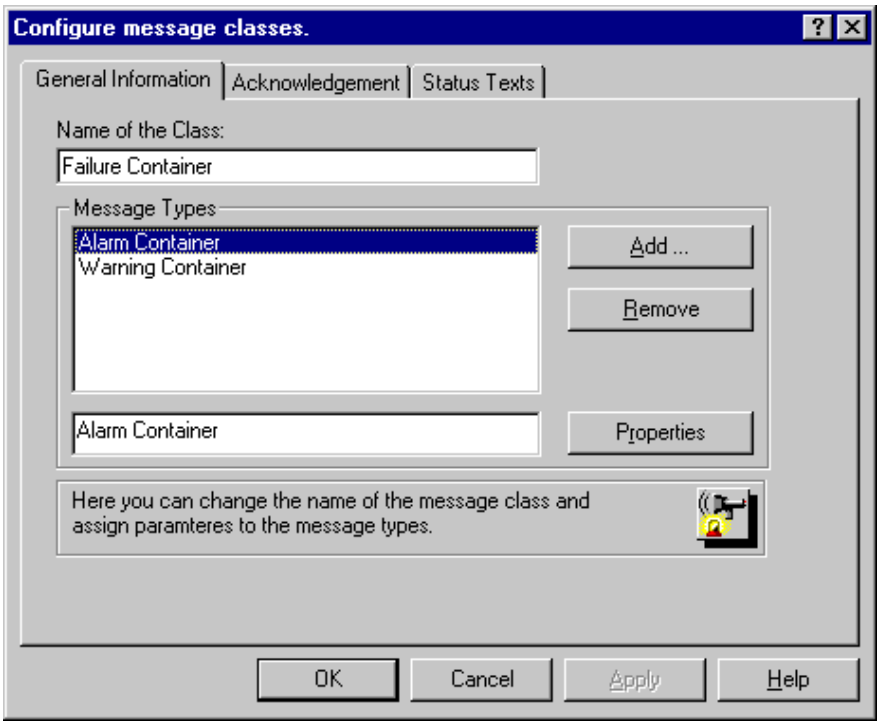
тип подтверждения

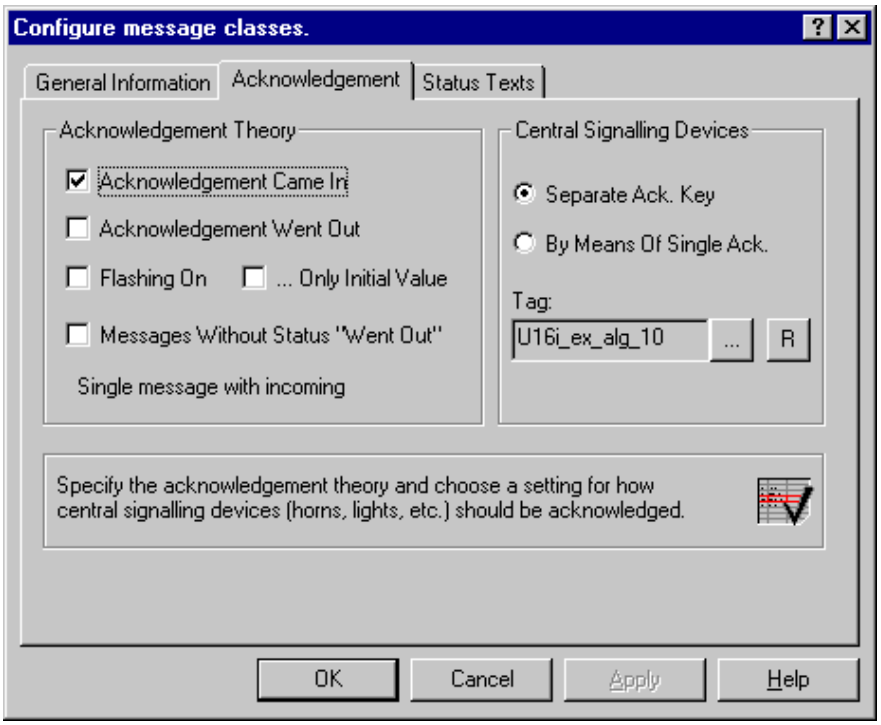
соответствующий текст состояния

звуковую/цветовую сигнализацию

Создание нового класса сообщений

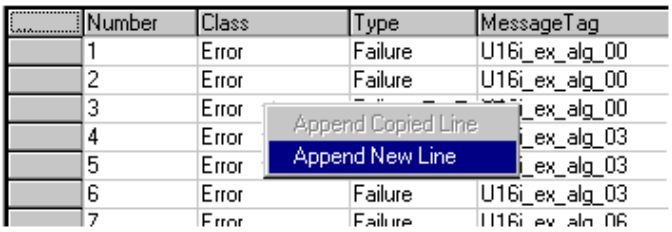
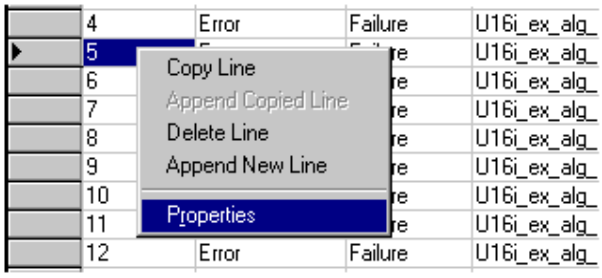
Шаг	Процедура: Создание нового класса сообщений
1	Щелкнув R (правой кнопкой мыши) на строке *Message Class (Класс сообщений)* откройте диалоговое окно *Add Message Class... (Добавление класса сообщений...)*. 
2	Добавление классов сообщений осуществляется с помощью кнопки –>. Диалоговое окно закрывается по нажатию на кнопку *OK*. 
3	По щелчку (мыши) на элементе *Message Class (Класс сообщений)* отображаются все созданные классы сообщений, даже те, что были добавлены только что. Щелкнув R (правой кнопкой мыши) на иконке класса сообщений можно открыть диалоговое окно *Configure Message Class (Конфигурирование класса сообщений)*. 

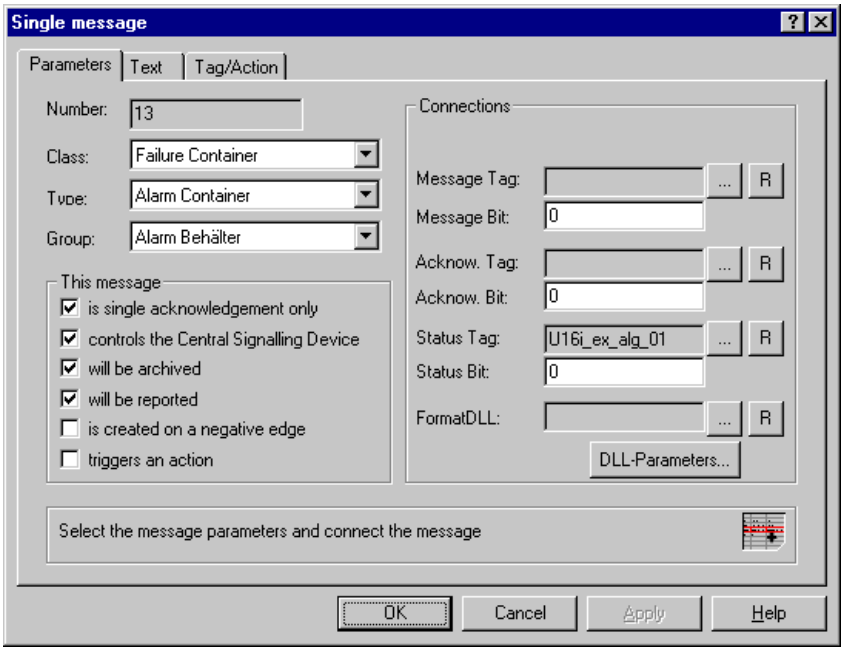
Шаг	Процедура: Создание нового класса сообщений
4	На закладке *General Information (Общая информация)* в качестве *Name of the Class (Названия класса)* введите *Container Error (Ошибка контейнера)*. С помощью кнопки *Add (Добавить)* можно вызвать диалоговое окно *Add Message Type... (Добавление типа сообщения...)*. Здесь с помощью кнопки –> перенесите из левого окна в правое два типа сообщения. Закройте диалоговое окно, нажав на *OK*. Если в поле *Message Types (Типы сообщений)* выбран один из новых типов сообщений, то диалоговое окно его свойств можно открыть по кнопке *Properties (Свойства)*. В качестве имени первого типа сообщений введите *Container Alarm (Аварийное сообщение контейнера)*. Цветовая схема конкретных сообщений выглядит следующим образом: Came in: Текст = черный, Фон = красный Went out: Текст = черный, Фон = зеленый Acknowledged: Текст = черный, Фон = оранжевый В качестве названия второго типа сообщений вводится *Container Warning (Предупредительное сообщение контейнера)*. Цветовая схема конкретных сообщений следующая: Came in: Текст = желтый, Фон = синий Went out: Текст = синий, Фон = RGB(207,163,146) Acknowledged: Text = белый, Фон = синий 

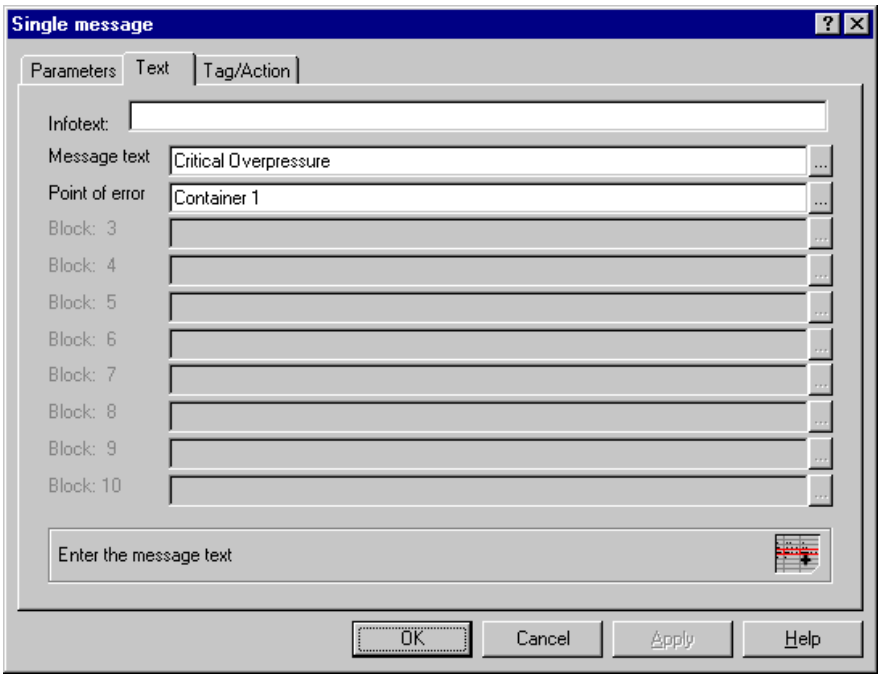
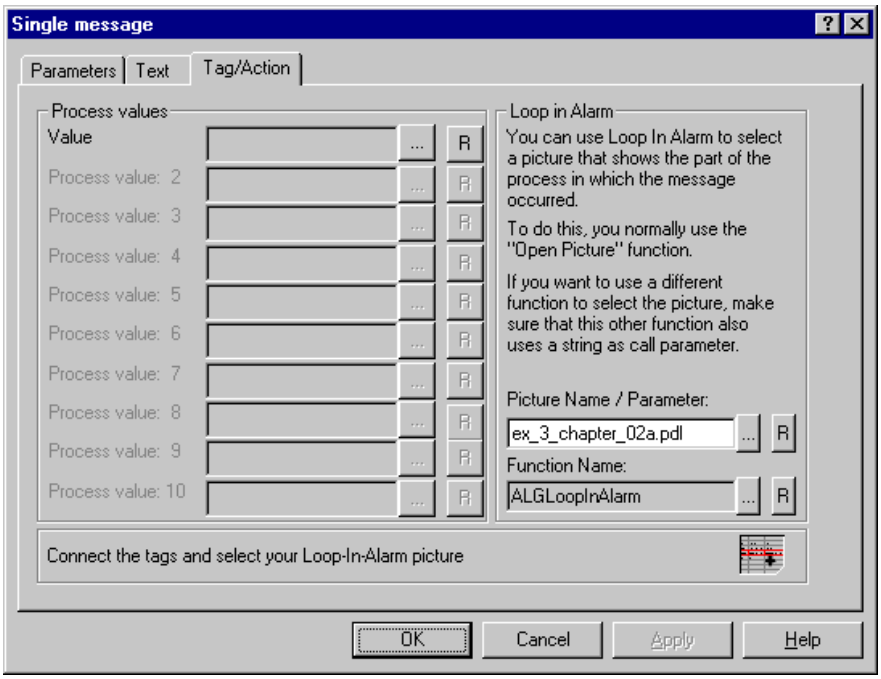
Шаг	Процедура: Создание нового класса сообщений
5	На закладке *Acknowledgment (Подтверждение)* в поле *Acknowledgment Theory (Способ подтверждения)* выберите флажок *Acknowledgment Came In (Подтверждение входа)*. В поле *Central Signalling Devices (Центральные сигнализирующие устройства)* установите опцию *Separate Acknowledgment Key (Отдельная клавиша подтверждения)*. В качестве *Tag (Тега)* укажите тег *U16i_ex_alg_10*. Этот тег контролирует центральный индикатор. Для подтверждения этого индикатора, на панели необходимо сконфигурировать отдельную кнопку. В случае стандартной панели это кнопка *Horn Acknowledgment (Подтверждение сигнала)*. 
6	На закладке *Status Texts (Тексты состояний)* не нужно делать никаких дополнительных настроек. Закройте диалоговое окно, нажав на кнопку *OK*.

4.2.3 Контроль по уставкам (продолжение)

Создание одиночных сообщений

Шаг	Процедура: Создание одиночных сообщений
1	Щелчком R (правой кнопки мыши) на окне таблицы добавьте 12 новых записей.  Выберите первую из только что добавленных записей (мышью). Щелчком R (правой кнопки мыши) на этой записи можно открыть диалоговое окно одиночных сообщений *Single Messages*. 

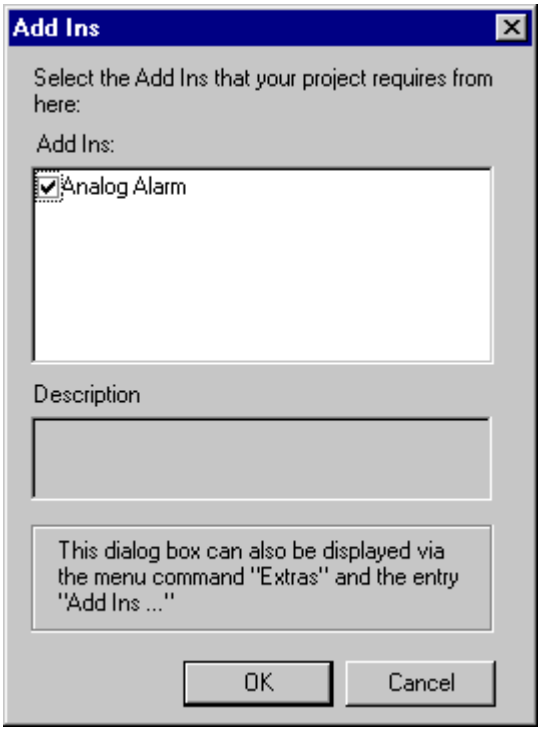
Шаг	Процедура: Создание одиночных сообщений
2	На закладке *Parameters (Параметры)* выберите класс сообщения *Container Error (Ошибка контейнера)* и тип сообщения *Container Alarm (Аварийное сообщение контейнера)*. В поле *This Message (Данное сообщение)* выберите переключатели *is single acknowledgment (одиночное подтверждение)*, *controls the horn (управляет сигналом)*, *will be archived (будет заархивировано)* и *will be reported (будет включено в отчет)*. В поле *Connections (Соединения)* в качестве *Status Tag (Тега состояния)* выберите тег *U16i_ex_alg_01*. В строке *Status Bit (Бит состояния)* введите *0*. *Event Tag (Тег события)* не устанавливается, поскольку сообщение генерируется контролем по уставкам. Аналогично не устанавливается и *Acknowledge Tag (Тег подтверждения)*. 

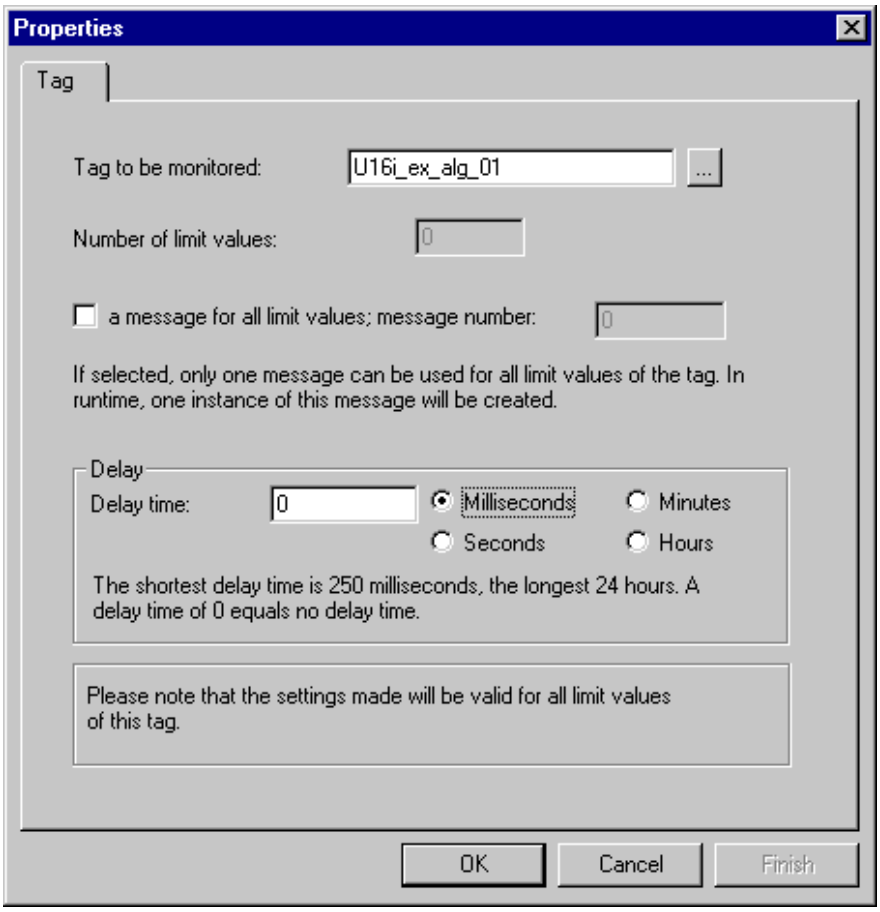
Шаг	Процедура: Создание одиночных сообщений
3	На закладке *Text (Текст)* в полях *Message Text (Текст сообщения)* и *Point of Error (Место ошибки)* введите значения *Critical Overpressure (Критическое давление)* и *Container 1 (Контейнер 1)* соответственно. В качестве *Infotext (Комментария)* введите *The pressure in container 1 has exceeded the critical value (Давление в контейнере 1 превысило критическое значение)*. 
4	На закладке *Tag/Action (Тег/Процедура)* можно установить тег для блока значений процесса *Value*. Однако если сообщение сгенерировано системой контроля по уставкам, то первый блок значений процесса в строке сообщения автоматически снабжается значением нарушенной уставки. Для подтверждения сделанных установок нажмите на *OK*. 

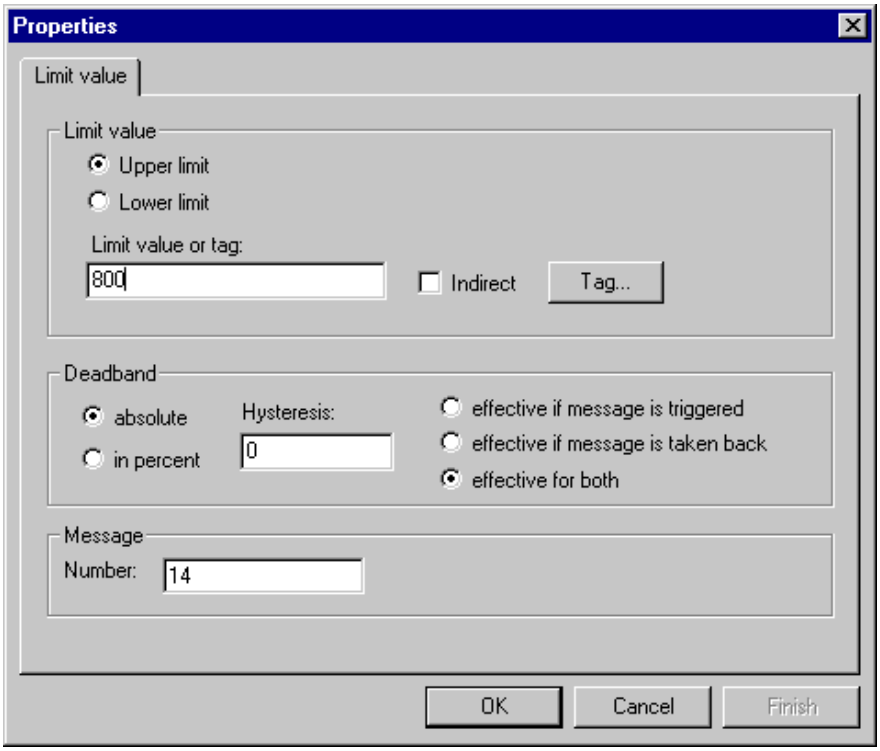
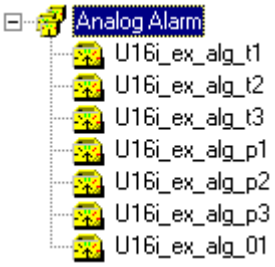
Шаг	Процедура: Создание одиночных сообщений
5	Только что созданное сообщение контролирует давление в первом из трех контейнеров. Для первого контейнера создаются еще три строки сообщений. Настройки делаются аналогично тому, как это описано в шаге 2, однако, для дополнительного сообщения типа *Container Error (Ошибка контейнера)* в поле *Message Text (Текст сообщения)* вводится значение *Critical Temperature (Критическая температура)* и соответствующим образом изменяется поле *Infotext (Комментарий)*. Кроме того, создаются два сообщения типа *Container Warning (Предупреждение контейнера)* и значениями *Pressure Warning (Предупреждение о давлении)* и *Temperature Warning (Предупреждение о температуре)*. Обратите внимание, что все переключатели в поле *This Message (Данное сообщение)* на закладке *Parameters (Параметры)* диалогового окна *Single Message (Одиночное сообщение)* отключены. Для всех сообщений, принадлежащих к контейнеру 1, используется один и тот же тег состояния, но с измененным битом состояния.
6	Для каждого из двух других контейнеров также создается по четыре сообщения. При этом *Status Tags (Теги состояний)* и тексты в поле *Point of Error (Место ошибки)* должны быть настроены соответствующим образом.

Number	Class	Type	MessageTag	MessageBit	Status tag	Status bit	Message text	Point of err
5	Error	Failure	U16i_ex_alg_03	1	U16i_ex_alg_05	1	Feedback Error	Motor 2
6	Error	Failure	U16i_ex_alg_03	2	U16i_ex_alg_05	2	Bimetal Error	Motor 2
7	Error	Failure	U16i_ex_alg_06	0	U16i_ex_alg_08	0	Lock Error	Motor 3
8	Error	Failure	U16i_ex_alg_06	1	U16i_ex_alg_08	1	Feedback Error	Motor 3
9	Error	Failure	U16i_ex_alg_06	2	U16i_ex_alg_08	2	Bimetal Error	Motor 3
10	Error	Failure	U16i_ex_alg_09	0	U16i_ex_alg_11	0	Lock Error	Motor 4
11	Error	Failure	U16i_ex_alg_09	1	U16i_ex_alg_11	1	Feedback Error	Motor 4
12	Error	Failure	U16i_ex_alg_09	2	U16i_ex_alg_11	2	Bimetal Error	Motor 4
13	Failure Containe	Alarm Contain		0	U16i_ex_alg_01	0	Critical Overpressure	Container 1
14	Failure Containe	Alarm Contain		0	U16i_ex_alg_01	1	Critical Temperature	Container 1
15	Failure Containe	Alarm Contain		0	U16i_ex_alg_04	0	Critical Overpressure	Container 2
16	Failure Containe	Alarm Contain		0	U16i_ex_alg_04	1	Critical Temperature	Container 2
17	Failure Containe	Alarm Contain		0	U16i_ex_alg_07	0	Critical Overpressure	Container 3
18	Failure Containe	Alarm Contain		0	U16i_ex_alg_07	1	Critical Temperature	Container 3
19	Failure Containe	Warning Cont		0	U16i_ex_alg_01	2	Warning pressure	Container 1
20	Failure Containe	Warning Cont		0	U16i_ex_alg_01	3	Warning temperature	Container 1
21	Failure Containe	Warning Cont		0	U16i_ex_alg_04	2	Warning pressure	Container 2
22	Failure Containe	Warning Cont		0	U16i_ex_alg_04	3	Warning temperature	Container 2
23	Failure Containe	Warning Cont		0	U16i_ex_alg_07	2	Warning pressure	Container 3
24	Failure Containe	Warning Cont		0	U16i_ex_alg_07	3	Warning temperature	Container 3

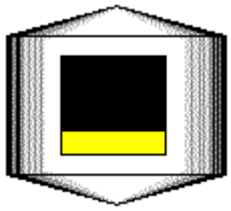
Конфигурация контроля по уставкам

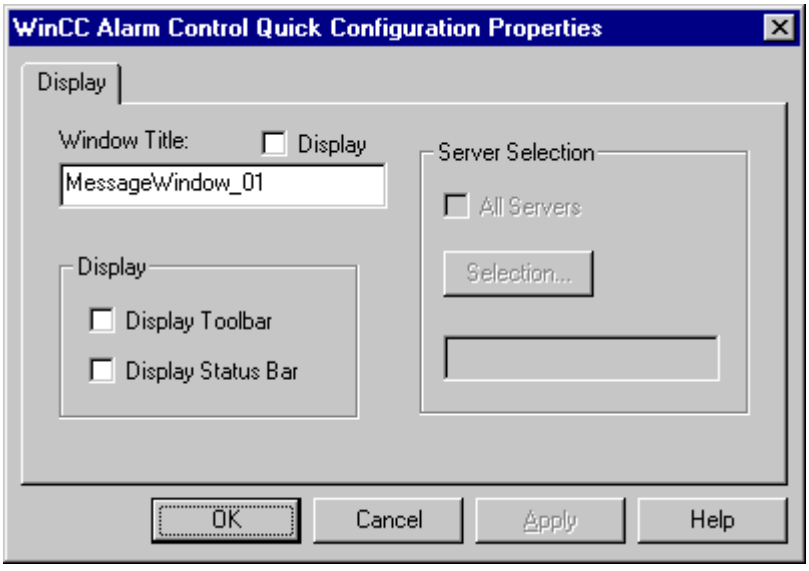
Шаг	Процедура: Конфигурация контроля по уставкам
1	Если *Limit Value Monitoring (Контроль по уставкам)* (Analog Alarm) не представлен в навигационном окне, его необходимо загрузить. Это делается командой меню *Options (Опции)* → *Add Ins (Дополнения)* в системе *Alarm Logging*. В отображенном диалоговом окне переключатель элемента для контроля по уставкам должен быть включен. 
2	Щелкнув R (правой кнопки мыши) на элементе *Limit Value Monitoring (Контроль по уставкам)* (Analog Alarm) и выбрав во всплывающем меню пункт *New (Создать)*, можно вызвать диалоговое окно *Properties (Свойства)* тега. В этом окне можно задать новый тег для контроля по уставкам. 

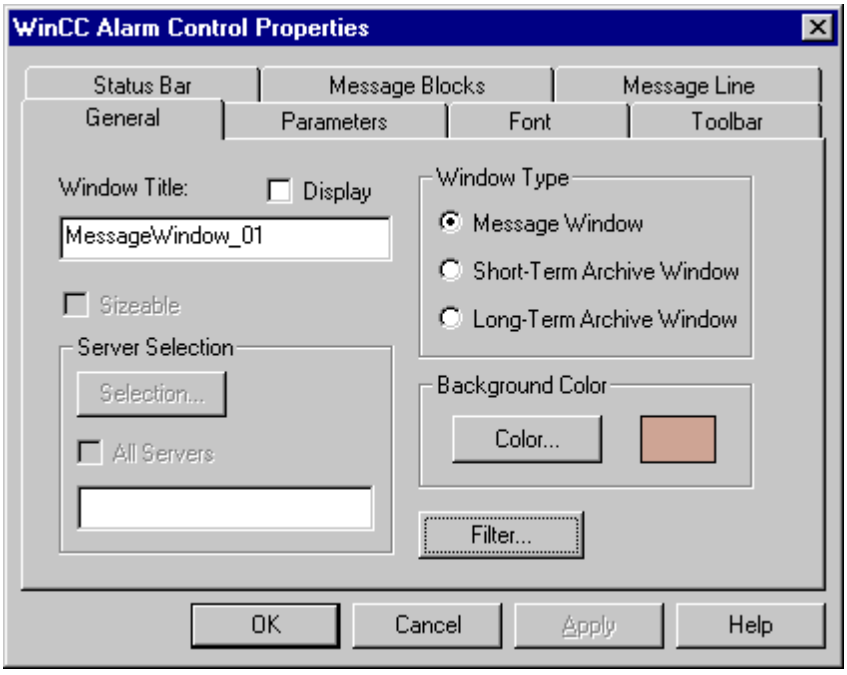
Шаг	Процедура: Конфигурация контроля по уставкам
3	В данном диалоговом окне в качестве *Tag to be monitored (Контролируемого тега)* установлен тег *U16i_ex_alg_t1*, содержащий температуру первого контейнера. Переключатель *a message for all limit values (сообщение для всех уставок)* не включен. В качестве *Delay Time (Время задержки)* оставьте *0*. Выход из диалогового окна осуществляется по нажатию на кнопку *OK*. 
4	В правом окне отображается иконка контролируемого тега. Щелкнув по ней R (правой кнопкой мыши) и выбрав во всплывающем меню пункт *New (Создать)* можно открыть диалоговое окно *Properties (Свойств)* новой уставки. В этом диалоговом окне тегу может быть назначена новая уставка. 

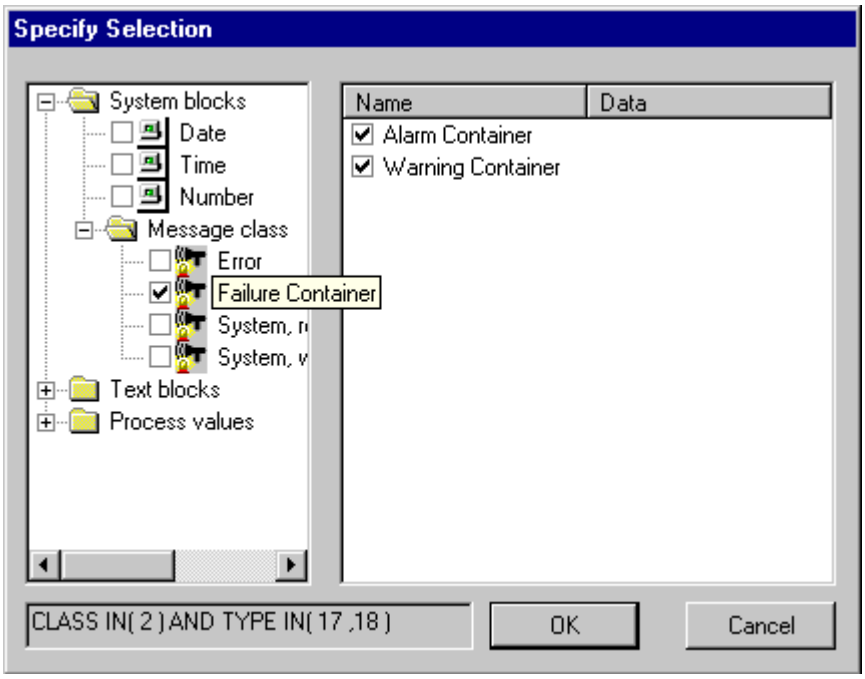
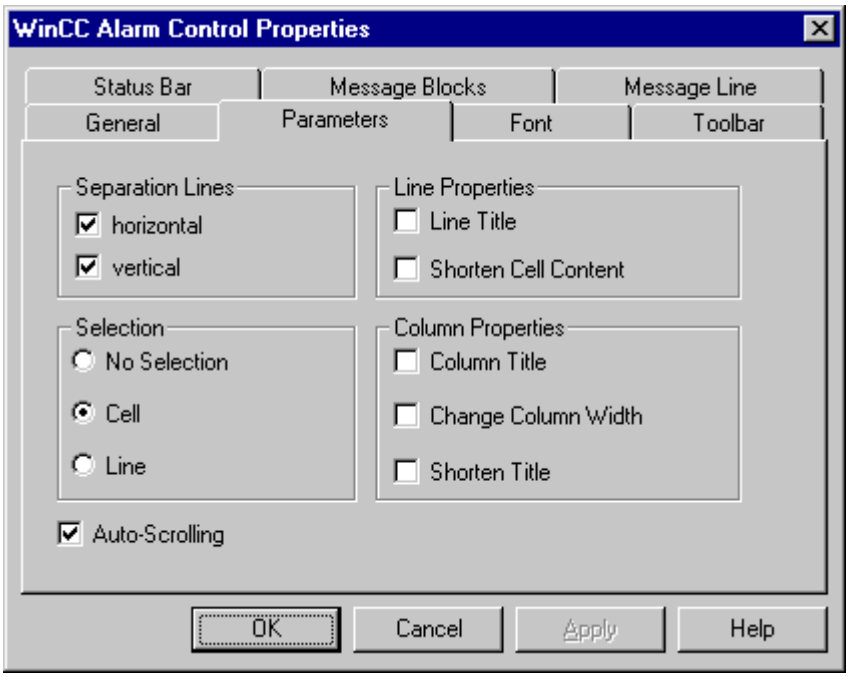
Шаг	Процедура: Конфигурация контроля по уставкам
5	На закладке *Limit Value (Уставка)* выберите опцию *Upper Limit (Верхний предел)*. В поле *Limit Value or Tag (Уставка или тег)* в качестве предельного значения введите *800*. В поле *Hysteresis (Гистерезис)* оставьте *0*. В качестве *Message (Сообщения)* в поле *Number (Число)* введите *14*. Это аварийное сообщение превышения допустимой температуры в первом контейнере. Выйти из диалогового окна можно, нажав на кнопку *OK*. Для того же тега определите вторую уставку. В поле *Limit Value (Уставка)* снова выберите опцию *Upper Limit (Верхняя уставка)*. Однако в поле *Limit Value or Tag (Уставка или тег)* введите *500*. В поле *Message (Сообщение)* в качестве *Number (Число)* введите *20*. Если будет наблюдаться превышение температуры в первом контейнере, то будет сгенерировано предупреждающее сообщение. 
6	Оставшиеся пять проверяемых тегов создаются так же, как описано в шагах 2 и 3, каждый с двумя сконфигурированными уставками. По D (двойному щелчку мыши) на знаке “+” перед элементом *Limit Value Monitoring (Контроль по уставкам)* раскроется список всех созданных тегов. 

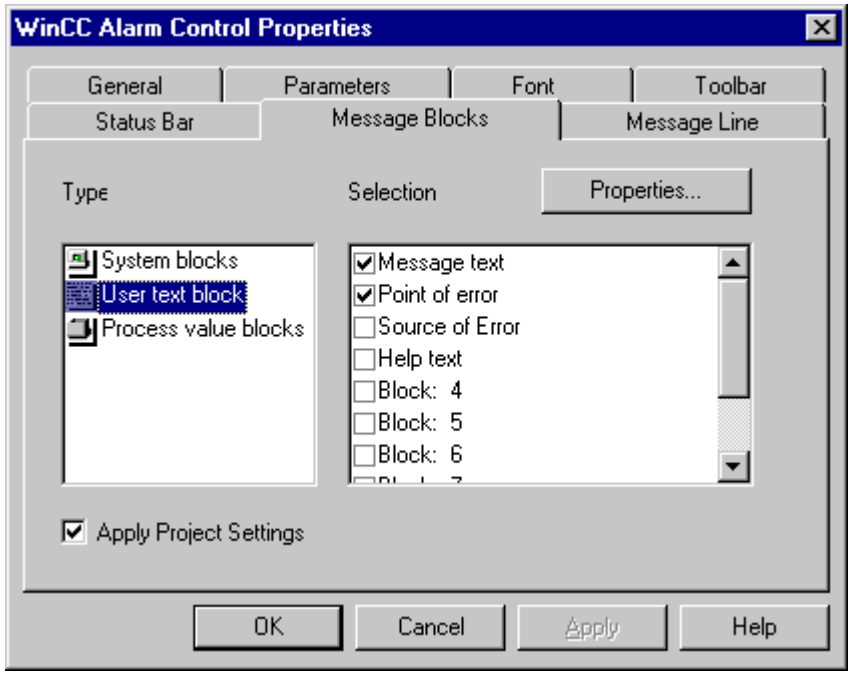
Реализация в графическом дизайнере

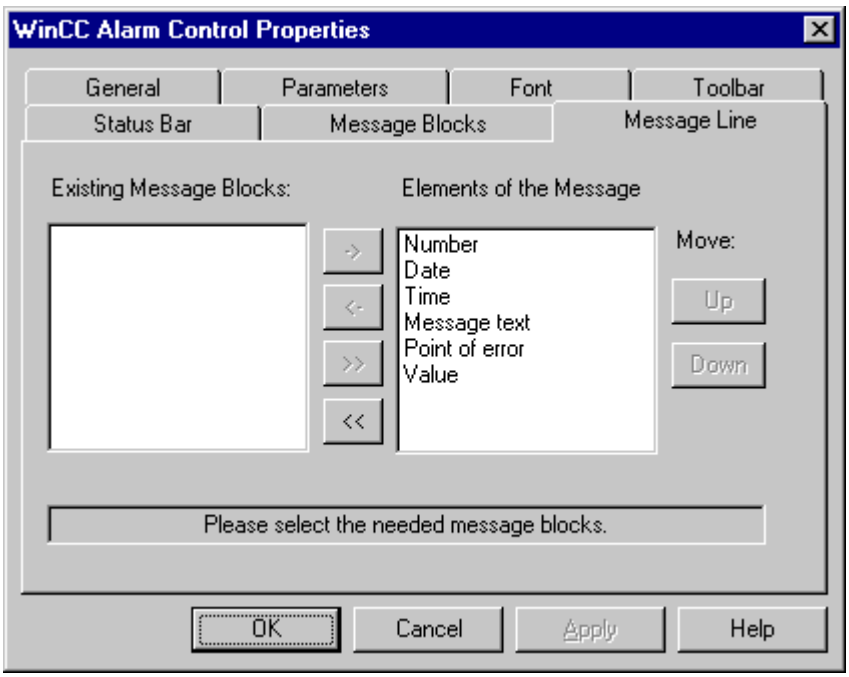
Шаг	Процедура: Реализация в графическом дизайнере
1	Моделирование каждого их контролируемых значений процесса осуществляется с помощью объекта *Windows Object (Объекты Windows)* → *Slider Object (Бегунок)*. В данном примере таких бегунков 6: от *Slider Object1* до *Slider Object6*. У *Slider Object1*создайте *прямое соединение* для *Events (События)* → *Property Topics (Разделы свойств)* → *Miscellaneous (Разное)* → *Process Driver Connection (Соединение с драйвером процесса)*, которое передает текущее значение бегунка тегу *U16i_ex_alg_t1*. Этот бегунок моделирует температуру в первом контейнере. Таким же образом сконфигурируйте бегунки для остальных тегов. Для согласования позиции бегунка с текущим значением тега при открытии кадра создается *процедура Си* для *Events (События)* → *Miscellaneous (Разное)* → *Open Picture (Открытие кадра)*.
2	Кроме того, каждому бегунку назначьте объект *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле ввода/вывода)*, показывающий текущее значение тега. В данном примере это объекты от *I/O Field1* до *I/O Field6*. Для *I/O Field1*создайте *соединение с тегом U16i_ex_alg_t1*. Это делается в поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Название кадра)*. Это *поле ввода/вывода*, назначенное первому бегунку. Таким же образом каждому из оставшихся бегунков назначьте соответствующее *поле ввода/вывода*.
3	Отображение конкретных контейнеров реализуется посредством *объекта Tank4* из стандартной библиотеки. В данном примере это объекты *Tank41*, *Tank42* и *Tank43*. Эти объекты используются только для отображения и не имеют динамических свойств. 
4	*Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)* назначается каждому контейнеру и служит для отображения предупреждающих световых сигналов. В данном примере это объекты от *Status Display1* до *Status Display3*. В данном примере для *Status Display1* в качестве *Basic Picture (Базового рисунка)* установлен *Blinker blinkt nicht.gif*, а в качестве *Flash Picture (Мигающего рисунка)* установлен *Bliker blinkt.gif*. Свойство *Flash Picture Active (Мигание рисунка активно)* в *Property (Свойство)* → *State (Состояние)* → *Flashing (Мигание)* установлено в *no (нет)*. Для того же самого свойства создается процедура Си, которая инициирует мигание в случае прихода аварийного сообщения от соответствующего контейнера. Два других индикатора состояния конфигурируются аналогично. 

Шаг	Процедура: Реализация в графическом дизайнере
5	Дополнительно конфигурируется *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*, который отображает сигнал. В данном случае это *Status Display4*. В качестве базового рисунка установлен *Hupe hupt nicht.gif*, а в качестве мигающего – *Hupe hupt.gif*. Свойство *Flash Picture Active (Мигание рисунка активно)* в *Property (Свойство)* → *State (Состояние)* → *Flashing (Мигание)* установлено в *no (нет)*. Для этого же атрибута создайте процедуру Си, которая инициирует мигание, если приходит аварийное сообщение, относящееся к одному из трех контейнеров, то есть, если тег в *Alarm Logging* для класса сообщений *Container Error*, управляющий центральным индикатором, перейдет в состояние *1*. В данном примере это тег *U16i_ex_alg_10*. Процедура Си, генерирующая аудио–сигналы при мигании объекта создается для атрибута *Properties (Свойства)* → *Geometry (Геометрия)* → *Width (Ширина)*. 
6	Для отображения сообщений, сконфигурированных в *Alarm Logging*, используется *WinCC Alarm Control*. Этот объект выбирается из палитры объектов и помещается в кадр.
7	После размещения данного элемента управления в кадре, его конфигурационное диалоговое окно отображается автоматически. В качестве *Window Title (Названия окна)* введите *MessageWindow_01*. Переключатель *Display (Отображение)* оставьте выключенным. В процедурах Си, создаваемых позже, этот заголовок окна используется для ссылки на соответствующий элемент управления. Переключатели *Toolbar (Панель инструментов)* and *Status Bar (Строка состояния)* выключены. Выход из конфигурационного диалогового окна осуществляется с помощью кнопки *OK*. 

Шаг	Процедура: Реализация в графическом дизайнере
8	Откройте диалоговое окно свойств элемента управления. Это окно отображается по DD (двойному щелчку мыши) на данном элементе управления. На закладке *General Information (Общая информация)* кнопка *Color (Цвет)* используется для согласования цвета фона с существующей в проекте цветовой схемой. Кнопка *Selection (Выбор)* используется для выбора одиночных сообщений, предварительно созданных в системе регистрации аварийных сообщений Alarm Logging для отображения данным элементом управления. 

Шаг	Процедура: Реализация в графическом дизайнере
9	По щелчку (мыши) на элементе *System Block Message Class (Классы сообщений системных блоков)* → *Container Error (Ошибка контейнера)* в правом окне высвечиваются 2 переключателя. Оба они включены. Это означает, что в режиме исполнения в окне сообщений будут отображаться сообщения только классом сообщения "ошибка контейнера". 
10	На закладке *Parameters (Параметры)* переключатели *Line Title (Заголовок строки)*, *Column Title (Заголовок столбца)* и *Change Column Width (Изменение ширины столбца)* выключены. В поле *Selection (Выбор)* выбрана опция *Cell (Ячейка)*. 

Шаг	Процедура: Реализация в графическом дизайнере
11	На закладке *Message Blocks (Блоки сообщений)* выберите столбец, который позднее будет отображен в строке сообщений. В данном случае в поле *Type (Тип)* с помощью (мыши) выбираются *System blocks (Системные блоки)*. В правом окне выбираются поля *Date (Дата)*, *Time (Время)* и *Number (Количество)*. Для элемента *User Text Blocks (Пользовательские текстовые блоки)* выбираются *Message Text (Текст сообщения)* и *Point of Error (Место ошибки)*. Для элемента *Process Value Blocks (Блоки значений процесса)* выбирается *Value (Значение)*. 

Шаг	Процедура: Реализация в графическом дизайнере
12	На закладке *Message Line (Строка сообщений)* предварительно выбранные *Message Blocks (Блоки сообщений)* назначается строке сообщений. В поле *Available Message Blocks* перечисляются все доступные столбцы. Нажатием на кнопку –> в строку можно добавить каждый блок сообщений по отдельности. При нажатии на кнопку >> все перечисленные в окне блоки сообщений добавляются в строку сообщений одновременно. Выход из диалогового окна свойств осуществляется с помощью кнопки *OK*. 
13	Для панели инструментов конфигурируется несколько объектов *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*, которые моделируют нажатие отдельных кнопок с помощью специальных функций.
14	Далее конфигурируется кнопка для подтверждения единичного сообщения. Эта кнопка также подтверждает сигнал, если тот был активизирован. Соответствующие стандартные функции: *AXC_OnBtnSinglAckn(lpszPictureName,lpszObjectName)* *AXC_OnBtnHornAckn(lpszPictureName,lpszObjectName)* 
15	Конфигурируются дополнительные кнопки: одна кнопка для группового подтверждения и одна кнопка для вызова диалогового окна Infotext (Информационный текст). Соответствующие стандартные функции: *AXC_OnBtnVisibleAckn(lpszPictureName,lpszObjectName)* *AXC_OnBtnInfo(lpszPictureName,lpszObjectName)* 

Шаг	Процедура: Реализация в графическом дизайнере
16	В качестве замены для кнопки, которая включает и выключает функцию авто скроллинга, используется *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*. В данном случае это объект *Status Display6*. Для атрибута *Properties (Свойства)* → *State (Состояние)* → *Current Status (Текущий статус)* создается *соединение с тегом BINi_ex_alg_00*. Этот тег содержит информацию о том, в каком состоянии находится режим авто скроллинга: во включенном или в выключенном. Для *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создается процедура Си, которая инвертирует тег *BINi_ex_alg_00* и вызывает стандартную функцию *AXC_OnBtnScroll(lpszPictureName,lpszObjectName)*. При открытии кадра тег *BINi_ex_alg_00* устанавливается в *0*, так как авто скроллинг меняет значение, если с окна сообщений снято выделение. 

Процедура Си для Status Display1

```
#include "apdefap.h"
 BOOL _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertу
{
WORD state;

state=GetTagWord("U16i_ex_alg_01");

if ((state&1)||(state&2)) return TRUE;
else return FALSE;
}
```

Чтение тега состояния первого контейнера. Если аварийное сообщение находится в состоянии ожидания, то свойству будет возвращено значение *TRUE* и замигает лампа сигнализации.

Данная *процедура Си* запускается при изменении тега состояния первого контейнера.

Процедура Си для Status Display4

```
#include "apdefap.h"
 BOOL _main(char* lpszPictureName, char* lpszObjectName, char* lpszPropertу
{
if (GetTagWord("U16i_ex_alg_10")&1) return TRUE;
else return FALSE;
}
```

Если центральный индикатор активизирован, то свойству будет присвоено значение *TRUE (Истинно)* и будет отображен оптический сигнал.

Данная *процедура Си* запускается при изменении тега, управляющего центральным индикатором.

Процедура Си для генерации звуковых сигналов

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
#pragma code ("winmm.dll")
BOOL PlaySound(LPCSTR pszSound,HMODULE hMod,DWORD fdwSound);
#define SND_FILENAME 0x00020000L
#define SND_ASYNC 0x0001
#pragma code ()

char szProjectName[MAX_PATH];
CMN_ERROR Error;
char szSoundFilePath[MAX_PATH] = "";
char szSoundFile[MAX_PATH] = "Hupe.wav";

if (GetFlashFlashPicture(lpszPictureName,lpszObjectName)) {
        if (DMGetRuntimeProject( szProjectName, MAX_PATH, &Error)) {
                strncat(szSoundFilePath,szProjectName,
                strlen(szProjectName)-strlen(strrchr(szProjectName,\\)+1));
        strcat(szSoundFilePath,szSoundFile);
        //MessageBeep((WORD)-1);
        PlaySound(szSoundFilePath,NULL,SND_FILENAME| SND_ASYNC);

        }
}

return 56;
}
```

Загрузка динамической библиотеки *winmm.dll*. Эта библиотека содержит функции воспроизведения звуковых файлов.

При мигании объекта *Status Display4* проигрывается файл *Hupe.wav*, который располагается в папке проекта. Для этого необходимо определить папку проекта с помощью функции *DMGetRuntimeProject* и путь для этого файла.

Вызов функции *PlaySound*.

Данная *процедура Си* выполняется в односекундном цикле.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Созданный класс сообщения должен быть настроен в соответствии с вашими требованиями.

Тип окна сообщений должен быть также настроен в соответствии с вашими требованиями.

4.2.4 Окно сообщений (ex_3_chapter_02b.pdl)

Постановка задачи

С помощью окна сообщений контролируется несколько процессов. Если приходит сообщение, то кнопка на панели инструментов должна давать возможность перехода к тому месту, где произошла ошибка.
Окно сообщений создается с использованием стандартных инструментальных средств системы *Alarm Logging*, используются стандартная панель инструментов и стандартная строка состояния.

Концепция реализации

Данный пример использует сообщения и кадры, созданные в предыдущих примерах. Необходима функция проекта, которая выполняет смену кадра, при нажатии кнопки *Loop In Alarm* на панели инструментов.
Окно сообщений создается в графическом дизайнере с использованием элемента управления *WinCC Alarm Control*. Никакие дополнительные объекты не нужны.

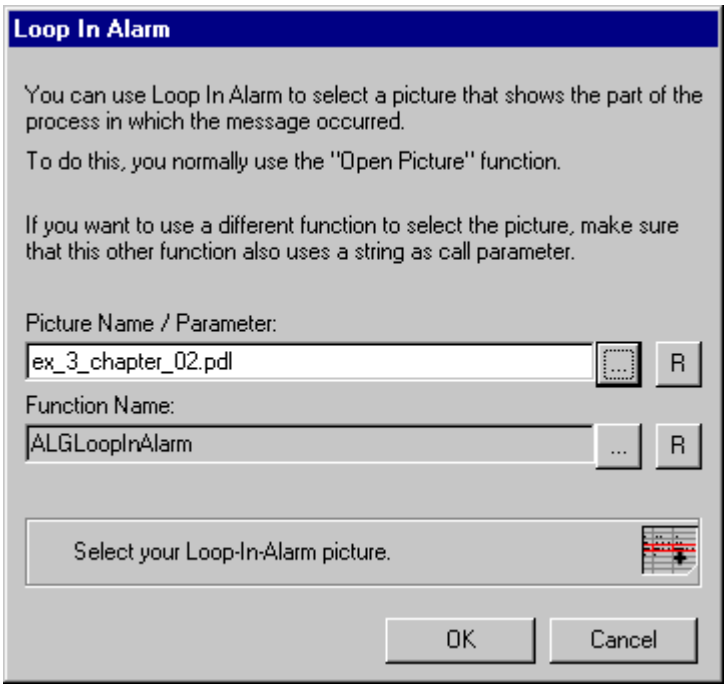
Замечание:

Настройки, сделанные в таблице *Configure Message Blocks (Конфигурирование блоков сообщений)* предыдущего примера, считаются завершенными, и в дальнейшем повторно описываться не будут.

Реализация примера

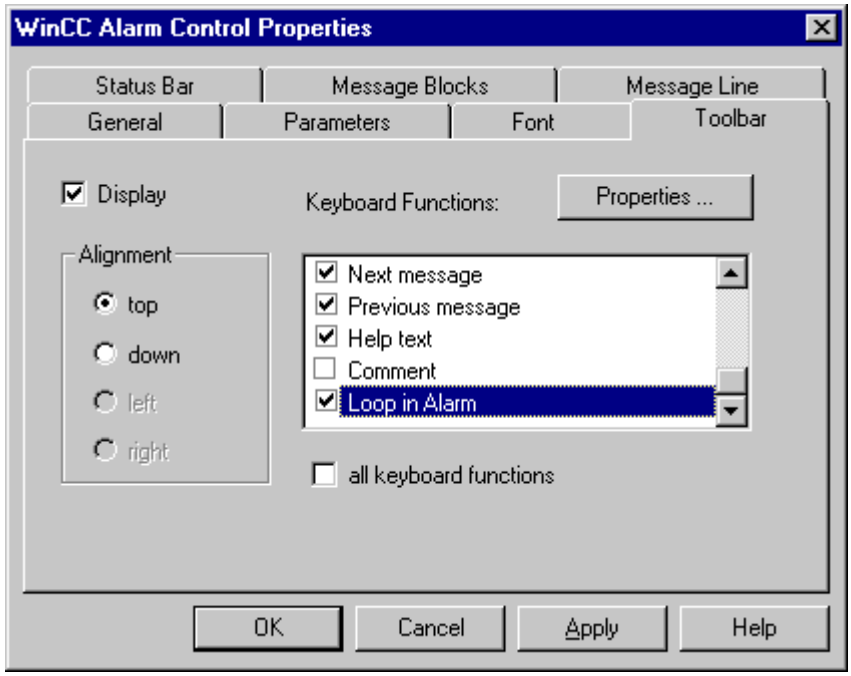
Шаг	Процедура: Реализация примера
1	В *проводнике WinCC* откройте редактор *Alarm Logging*.
2	Для каждого одиночного сообщения устанавливается *Loop in Alarm (Цикл аварийного сообщения)*. Эта функция позволяет напрямую менять текущий кадр соответствующим кадром сообщения. В качестве функции смены кадра по умолчанию установлена функция *OpenPicture*. В данном примере создается отдельная функция, выполняющую смену кадра в окне кадров. Параметры вызова этой функции предопределены системой *Alarm Logging*. Такая функция *ALGLoopInAlarm* создается в редакторе глобальных сценариев.
3	В окне таблицы *Alarm Logging* после D (двойного щелчка мыши) на столбце *Loop in Alarm* откроется соответствующее диалоговое окно выбранного одиночного сообщения.

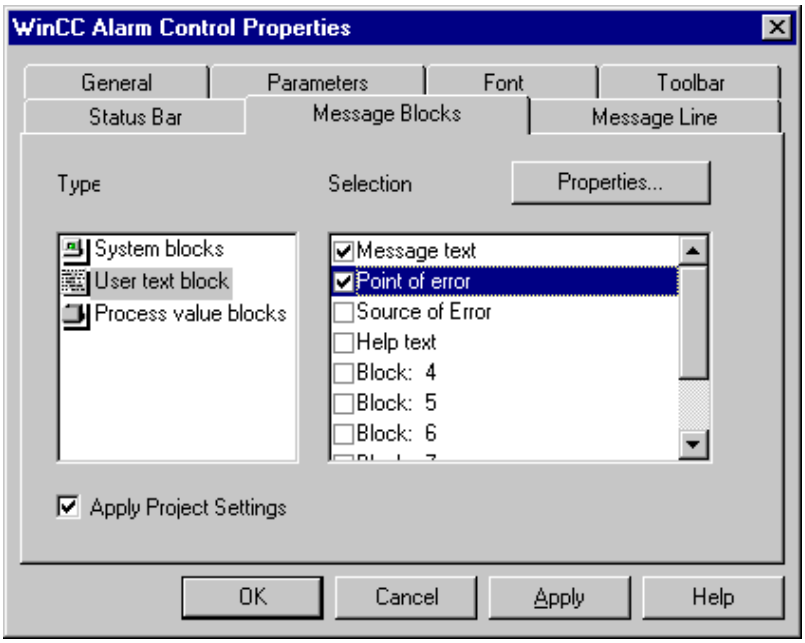
Acknowledgement bit	Loop in Alarm	Group
0	Not set	
0	Not set	
0	Not set	
0	Not set	
0	Not set	
0	Not set	
0	Not set	
0	Not set	

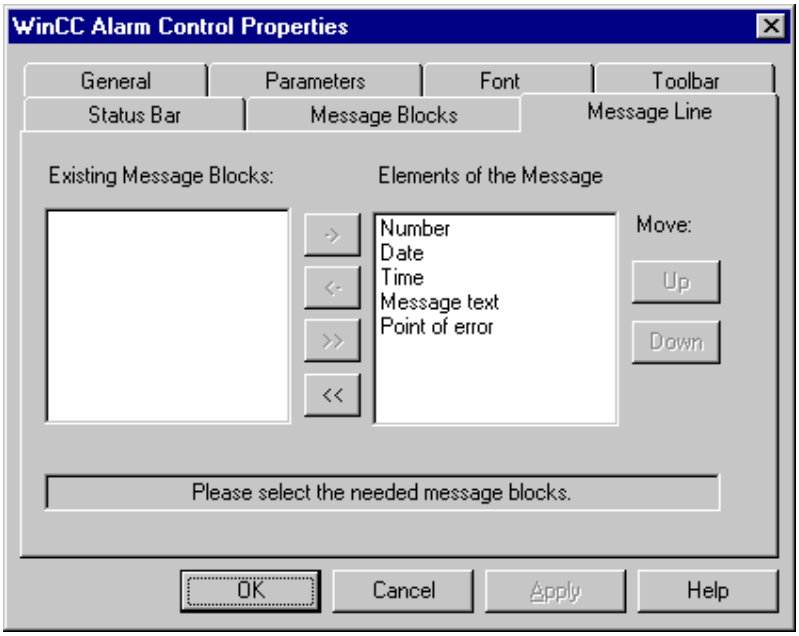
Шаг	Процедура: Реализация примера
4	В поле *Function Name (Имя функции)* задается функция *ALGLoopInAlarm*. Для сообщений, которые ссылаются на двигатели первого примера, в качестве *Picture Name (Названия кадра)/Call Parameter (Параметра вызова)* используется кадр *ex_3_chapter_02.pdl*, для сообщений второго примера используется кадр *ex_3_chapter_02a.pdl*. 
5	Конфигурирование функции *Loop in Alarm* может также выполняться на закладке *Tag/Action (Тег/Процедура)* диалогового окна свойств одиночного сообщения в поле *Loop in Alarm*. Настройки, сделанные в *Alarm Logging* сохраняются.

Реализация в графическом дизайнере

Шаг	Процедура:
1	Откройте *графический дизайнер* и создайте новый кадр. В данном примере это кадр *ex_3_chapter_02b.pdl*.
2	Для отображения сообщений, сконфигурированных в *Alarm Logging*, используется объект *WinCC Alarm Control*. Этот объект выбирается из палитры объектов и помещается в кадр.
3	После размещения элемента управления в кадре конфигурационное диалоговое окно будет отображено автоматически. Закрытие конфигурационного диалогового окна осуществляется нажатием на кнопку *OK*. При помощи D (двойного щелчка мыши) на элементе управления откройте его диалоговое окно *Properties (Свойства)*. Все настройки можно сделать на закладке *General Information (Общая информация)*.

Шаг	Процедура:
4	На закладке *Toolbar (Панель инструментов)* должны быть включены следующие переключатели: Single Acknowledgment (Одиночное подтверждение) Group Acknowledgment (Групповое подтверждение) Auto–Scroll On/Off (Авто скроллинг Вкл/Выкл) Report Functions (Функции отчетов) Beginning of the List (Начало списка) End of List (Конец списка) Next Message (Следующее сообщение) Previous Message (Предыдущее сообщение) Infotext (Комментарий) Loop in Alarm (Цикл аварийного сообщения) 

Шаг	Процедура:
5	На закладке *Message Blocks (Блоки сообщений)* выбираются столбцы, которые позднее будут отображены в строке сообщений. В данном примере в поле *Type (Тип)* с помощью (мыши) выбираются *system blocks (системные блоки)*. В правом окне выбираются *Date (Дата)*, *Time (Время)* и *Number (Количество)*. Для элемента *User Text Blocks (Пользовательские текстовые блоки)* включаются *Message Text (Текст сообщения)* и *Point of Error (Место ошибки)*. 

Шаг	Процедура:
6	На закладке *Message Line (Строка сообщений)* предварительно выбранные *Message Blocks (Блоки сообщений)* назначается строке сообщений. В поле *Available Message Blocks (Доступные блоки сообщений)* перечислены все доступные столбцы. Нажатием на кнопку –> в строку сообщений можно добавить каждый блок сообщений по отдельности. При нажатии на кнопку >> все блоки сообщений, перечисленные в окне, могут быть добавлены в строку сообщений одновременно. Выход из диалогового окна свойств выполняется по кнопке *OK*. 

Функция проекта ALGLoopInAlarm

```
void ALGLoopInAlarm(char* PictureName)
{

SetPictureName("ex_0_startpicture_00.pdl","workspace", PictureName);

}
```

Вызов функции *SetPictureName* для смены кадра. Эта функция не может использоваться непосредственно в *Alarm Logging*, так как количество и тип ее вызываемых параметров не согласуются с указанными.

Замечание:

В панели инструментов *WinCC Alarm Control* существует кнопка для функций отчетов. Реализация отчета последовательности сообщений и его активизация описывается в примере Message Sequence Report (Отчет последовательности сообщений) (ex_3_chapter_02b.pdl) главы *Report Designer (Дизайнер отчетов)*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Функции *Loop in Alarm*, сконфигурированные для конкретных сообщений, должны быть настроены в соответствии с вашими требованиями.

Тип отображения окна сообщений должен быть настроен в соответствии с вашими требованиями.

4.2.5 Архивация сообщений (ex_3_chapter_02c.pdl)

Постановка задачи

Создаваться краткосрочный архив для 200 сообщений. Все сообщения должны отображаться в окне сообщений.

Окно сообщений должно управляться определяемой пользователем панелью инструментов. Эта панель инструментов должна содержать две специальные кнопки, которые позволяют пользователю отображать сообщения примера 1 или примера 2.

Концепция реализации

Данный пример использует сообщения, созданные в предыдущих примерах. Дополнительно к этому конфигурируется архив сообщений.

Окно сообщений создается в графическом дизайнере с помощью *WinCC Alarm Control*.

Панель инструментов реализуется с помощью нескольких объектов *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)*, *Smart Objects (Интеллектуальные объекты)* → *Status Displays (Индикаторы состояния)* и *Smart Objects (Интеллектуальные объекты)* → *Graphic Objects (Графические объекты)*.

Необходима функция проекта, которая осуществляет выбор в окне сообщений при нажатии кнопок выбора.

Создание необходимых тегов

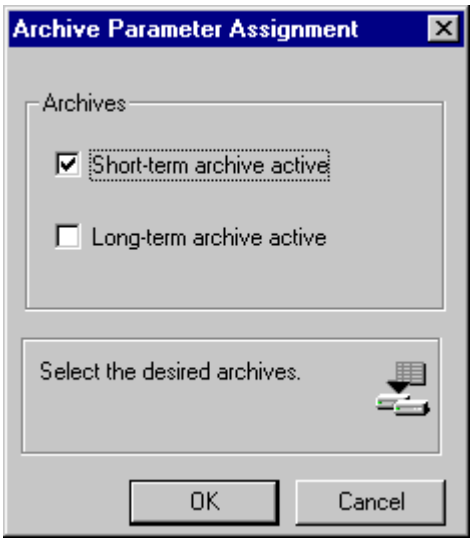
Шаг	Процедура: Создание необходимых тегов
1	Создаются три тега типа *Binary Tag (Двоичный тег)*. В данном случае это теги *BINi_ex_alg_00*, *BINi_ex_alg_01* и *BINi_ex_alg_02*.

Замечание:

Настройки, сделанные в первом и втором примерах, считаются завершенными. Они не будут объясняться повторно, однако, данный пример базируется на них.

Реализация в системе регистрации аварийных сообщений

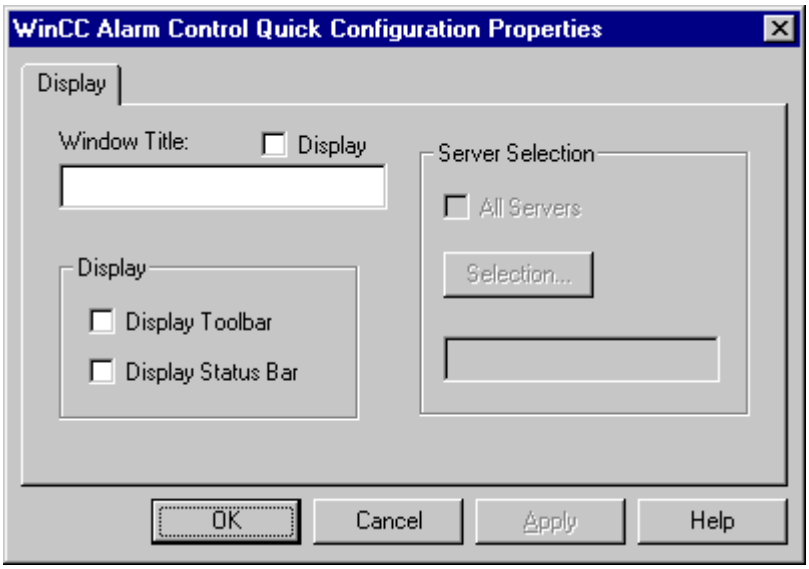
Шаг	Процедура: Реализация в системе регистрации аварийных сообщений
1	Из *проводника WinCC* откройте редактор *Alarm Logging*.

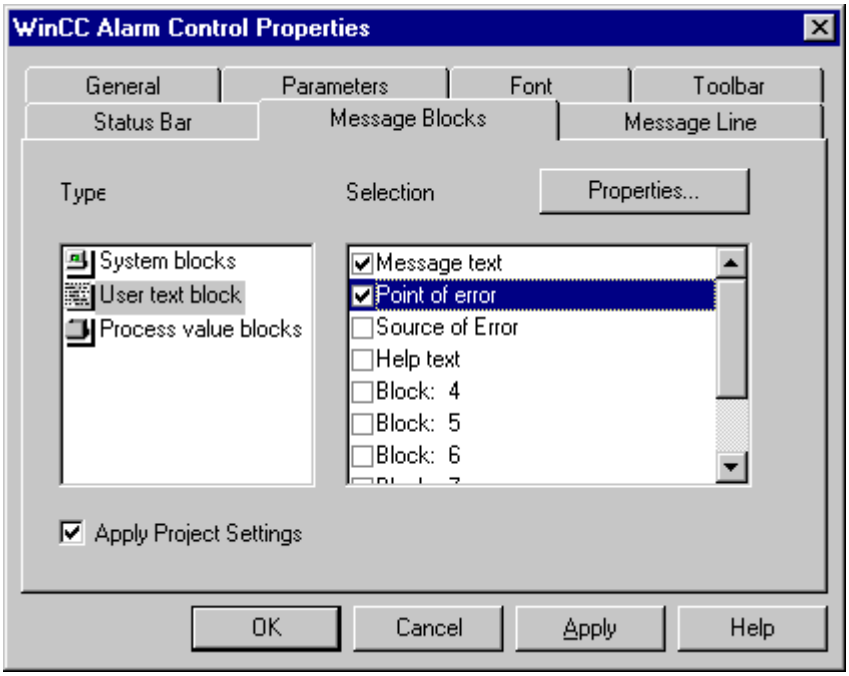
Шаг	Процедура: Реализация в системе регистрации аварийных сообщений
2	По щелчку R (правой кнопки мыши) на элементе *Archives (Архивы)*, откроется диалоговое окно *Archive Parameters (Параметров архива)*. 
3	В этом диалоговом окне выбирается переключатель *Short–term archive active (Активен краткосрочный архив)*. 
4	В правом окне отображается иконка краткосрочного архива. С помощью щелчка R (правой кнопки мыши) на этой иконке открывается диалоговое окно свойств краткосрочного архива. 

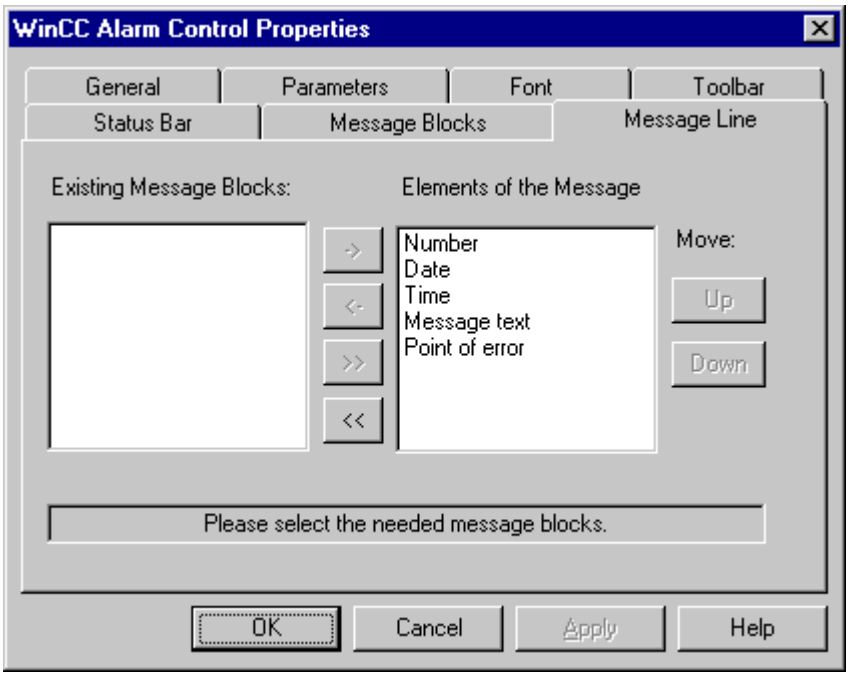
Шаг	Процедура: Реализация в системе регистрации аварийных сообщений
5	Архив должен быть сохранен на диске. В поле *Number of Entries (Количество записей)* вводится *200*. 

Реализация в графическом дизайнере

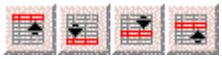
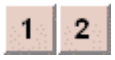
Шаг	Процедура: Реализация в графическом дизайнере
1	Откройте графический дизайнер и создайте новый кадр. В данном случае это кадр *ex_3_chapter_02c.pdl*.

Шаг	Процедура: Реализация в графическом дизайнере
2	После добавления элемента управления в кадр, его конфигурационное диалоговое окно будет отображено автоматически. В поле *Window Title (Заголовок окна)* вводится значение *MessageWindow_04*. Переключатель *Display (Отображение)* остается выключенным. В *процедуре Cu*, созданной позднее, этот заголовок окна используется для ссылки на соответствующий элемент управления. Переключатели *Toolbar (Панель инструментов)* и *Status Bar (Строка состояния)* отключены. Выйти из конфигурационного диалога можно по нажатию на кнопку *OK*. 
3	Откройте диалоговое окно свойств элемента управления D (двойным щелчком мыши) на данном элементе. Все настройки можно сделать на закладке *General Information (Общая информация)*. Поскольку отображаются все одиночные сообщения, выбора делать не нужно.

Шаг	Процедура: Реализация в графическом дизайнере
4	На закладке *Message Blocks (Блоки сообщений)* выбираются столбцы, которые будут позже отображены в строке сообщений. Например, в поле *Type (Тип)* с помощью (мыши) выбираются *system blocks (системные блоки)*. В правом окне включаются *Date*, *Time* и *Number*. Для элемента *User Text Blocks (Пользовательские текстовые блоки)* выбираются *Message Text (Текст сообщения)* и *Point of Error (Место ошибки)*. 

Шаг	Процедура: Реализация в графическом дизайнере
5	На закладке *Message Line (Строка сообщений)* предварительно выбранные *Message Blocks (Блоки сообщений)* назначается строке сообщений. В поле *Available Message Blocks (Доступные блоки сообщений)* перечислены доступные столбцы. Нажатием на кнопку –> в строку сообщений можно добавить каждый блок сообщений по отдельности. При нажатии на кнопку >> все блоки сообщений, перечисленные в окне, могут быть добавлены в строку сообщений одновременно. Выход из диалогового окна свойств осуществляется по кнопке *OK*. 
6	Для панели инструментов конфигурируется несколько *Windows Objects (Объектов Windows)* → *Buttons (Кнопок)*, которые имитируют нажатие отдельных кнопок с помощью стандартных функций.
7	Конфигурируются кнопка вызова диалогового окна выбора и кнопка для вызова диалогового окна комментариев (Infotext). Соответствующие стандартные функции: *ACX_OnBtnInfo()ACX_OnBtnSelect()* 

Шаг	Процедура: Реализация в графическом дизайнере
8	Как замена для кнопки, которая включает и выключает функцию авто скроллинга, используется *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*. В данном случае это объект *Status Display3*. Для атрибута *Properties (Свойства)* → *State (Состояние)* → *Current Status (Текущий статус)* создается *соединение с тегом BINi_ex_alg_00*. Этот тег содержит информацию о том, в каком состоянии находится режим авто скроллинга: Вкл. или Выкл. Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создается *процедура Cu*, которая инвертирует тег *BINi_ex_alg_00* и вызывает стандартную функцию *ACX_OnBtnScroll()*. При открытии кадра тег *BINi_ex_alg_00* устанавливается в *0*, так как авто скроллинг меняет значение, если с окна сообщений снято выделение. 

Шаг	Процедура: Реализация в графическом дизайнере
9	Если режим авто скроллинга выключен, то передвижение в окне сообщений выполняется с помощью четырех специальных кнопок. Эти кнопки заменяют соответствующие кнопки стандартной панели инструментов со следующими функциями: *ACX_OnBtnMsgFirst()ACX_OnBtnMsgLast()ACX_OnBtnMsgNext()ACX_OnBtnMsgPrev()* Эти кнопки делаются неактивными с помощью *Smart Object (Интеллектуальный объект)* → *Graphic Object (Графический объект)*, который помещается поверх них, если авто скроллинг включен. Это делается посредством *соединения с тегом BINi_ex_alg_00* в поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Display (Отображение)*. 
10	С помощью двух объектов *Smart Objects (Интеллектуальные объекты)* → *Status Displays (Индикаторы состояний)* реализуется переключение между типами отображений *Message Window (Окно сообщений)* и *Short–Term Archive Window (Окно краткосрочного архива)*. Текущее состояние окна сообщений сохраняется в теге *BINi_ex_alg_01*, который должен быть установлен в ноль при открытии кадра, так как окно сообщений при повторном открытии отображается как *окно краткосрочного архива*. У *Status Display1* для *Properties (Свойства)* → *State (Состояние)* → *Current Status (Текущий статус)* создайте связь с тегом *BINi_ex_alg_01*. Используя *Properties (Свойства)* → *Miscellaneous (Разное)* → *Operator–Control Enable (Разрешение управления оператором)* создайте *динамический диалог*, который делает объект активным только тогда, когда окно сообщений отображает краткосрочный архив, то есть тег *BINi_ex_alg_01* находится в состоянии 0. Для события *Events (События)* → *Mouse (Мышь)* → *Press Left (Нажатие левой кнопки)* создайте *процедуру Cu*, которая имитирует нажатие соответствующей кнопки на панели инструментов и инвертирует тег *BINI_EX_ALG_01*. Объект *Status Display2* конфигурируется таким же образом. Используются следующие стандартные функции: *ACX_OnBtnMsgWin()ACX_OnBtnArcShortt()* 
11	С помощью двух дополнительных объектов *Windows Objects (Объекты Windows)* → *Buttons (Кнопки)* в окне сообщений производится непосредственный выбор. Выбор может производиться для того, чтобы просмотреть сообщения, относящиеся к двигателям или контейнерам. Выбор выполняется функцией проекта, созданной в редакторе глобальных сценариев. Этой функции передаются номера сообщений, между которыми находятся отображенные сообщения. В данном случае это функция *SetMsgNrSelection*. 

Функция проекта для настройки выборки

```
BOOL SetMsgNrSelection(DWORD dwFrom, DWORD dwTo, LPSTR MsgTem)
{

   PCMN_ERROR          pError;
   BOOL                fRet;
   MSG_FILTER_STRUCT   Filter;

   memset(&Filter, \0,  sizeof( MSG_FILTER_STRUCT ) );
   strcpy( Filter.szFilterName, MsgTem);
   Filter.dwFilter = MSG_FILTER_NR_FROM|MSG_FILTER_NR_TO;
   Filter.dwMsgNr[0] = dwFrom;
   Filter.dwMsgNr[1] = dwTo;

   fRet = MSRTSetMsgWinFilter( &Filter, pError );

   if (fRet == FALSE)
   {
        printf("Error MSRTSetMsgWinFilter\r\n" );
        return FALSE;
   }
   else
        return TRUE;

}
```

Резервирование памяти для созданной структуры фильтров.

Присвоение значений элементу структуры фильтра, относящихся к данному приложению. В качестве *szFilterName* должно использоваться имя шаблона окна сообщений, на которое ссылается фильтр. В массив *dwMsgNr* вводятся начальное и конечное значения номеров сообщений, которые будут выбраны. При вызове функции эти значения являются передаваемыми параметрами. Флаг *dwFilter* установлен таким образом, что он идентифицирует структуру фильтра по номеру.

Вызов функции API *MSRTSetMsgWinFilter*, которая применяет созданный фильтр к выбранному шаблону окна сообщений.

Замечания относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Тип отображения окна сообщений должен быть настроен в соответствии с вашими требованиями.

Вид и элементы панели инструментов должны быть настроены в соответствии с вашими требованиями.

4.2.6 Групповые сообщения (ex_8_generator_00.pdl)

В режиме исполнения пример, относящийся к данной теме, выбирается щелчком (мыши) на изображенной выше кнопке. С помощью флажка *active (активный)* в данном кадре можно включить генератор сообщений. Он генерирует различные сообщения с интервалом в 10 секунд.

Постановка задачи

В кадре отображать предупреждения, сообщающие о присутствии сообщения определенного типа.
Эти сообщения уже были сконфигурированы в главах Битовая процедура сообщения (ex_3_chapter_02.pdl) и Контроль по уставкам (продолжение) и применяются в данном примере. Необходимо указать ожидающие предупреждения и аварийные сообщения в кадре контейнера и ошибки, происходящие в кадре двигателя. Аварийное сообщение имеет приоритет перед отказом и ошибкой. Если сообщение находится в состоянии ожидания, то переход к соответствующему кадру осуществляется с помощью кнопки.

Концепция реализации

Проверяемые одиночные сообщения объединяются в групповое сообщение. При генерации одиночного сообщения групповое сообщение генерируется также. Данному групповому сообщению назначается тег состояния и бит состояния.
Используя *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)* оценивается текущее состояние этого тега и отображается соответствующий символ.

Замечание:
Настройки, сделанные в первом и втором примерах, считаются завершенными, и повторно объясняться не будут.

Создание необходимых тегов

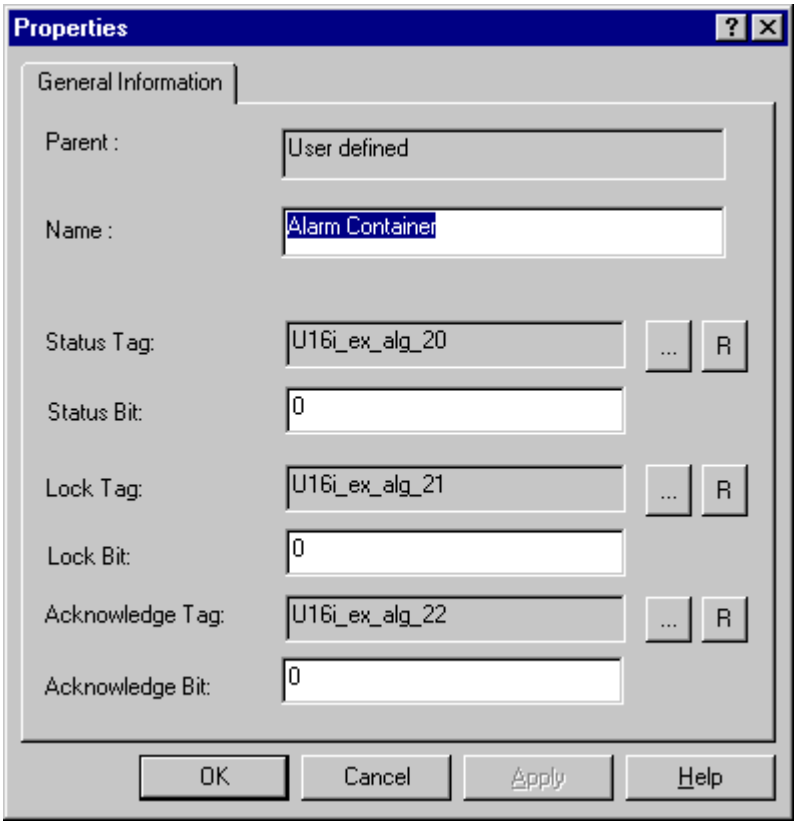
Шаг	Процедура: Создание необходимых тегов
1	Создание трех тегов типа *Unsigned 16–Bit Value (16–битная величина без знака)* в *менеджере тегов*. В данном примере это теги *U16i_ex_alg_20*, *U16i_ex_alg_21* и *U16i_ex_alg_22*. Они служат в качестве тегов состояния, блокировки и подтверждения.

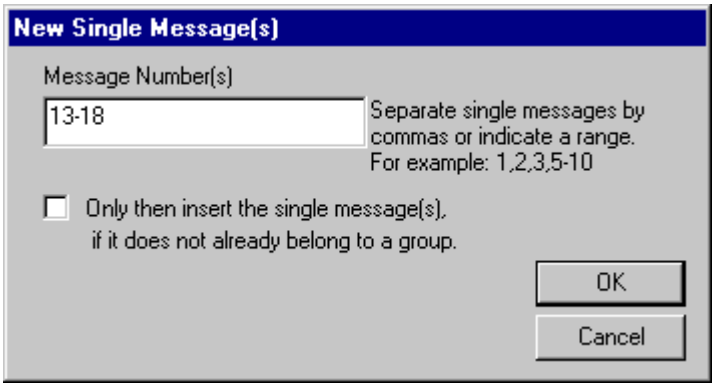
Общая информация

Если создан новый класс сообщений, то групповое сообщение для этого класса сообщений создается автоматически. Все сообщения в пределах этого класса сообщений переносятся в групповое сообщение. Свойства классов сообщений и типов сообщений в групповом сообщении могут быть изменены независимо и поэтому могут быть связаны с различными тегами состояния, блокировки и подтверждения.
В данном примере, однако, подразумевается, что в проекте существуют и другие кадры, использующие те же самые классы сообщений. Это означает невозможность использования автоматически сгенерированных групповых

сообщений, так как соответствующее групповое сообщение должно также идентифицировать кадр, в котором оно произошло.
Следовательно, должны быть созданы определяемые пользователем групповые сообщения.

Создание новых групповых сообщений

Шаг	Процедура: Создание новых групповых сообщений
1	Откройте редактор *Alarm Logging*. При помощи D (двойного щелчка мыши) на строке *Group Messages (Групповые сообщения)* разверните две подстроки. Это будут элементы *Message Class (Класс сообщения)* и *User–Defined (Определенный пользователем)*. Щелчком R (правой кнопки мыши) на элементе *User–Defined (Определенный пользователем)* можно вызвать окно *New Group Message (Новое групповое сообщение)*. 
2	В отображенном диалоговом окне в поле *Name (Название)* вводится *Alarm Container (Аварийное сообщение контейнера)*. В качестве тегов *Status (Состояние)*, *Lock (Блокировка)* и *Acknowledge (Подтверждение)* выбираются предварительно созданные теги. В качестве номера бита всегда используется *0*. Выход из диалога осуществляется по нажатию на *OK*. 

Шаг	Процедура: Создание новых групповых сообщений
3	Аналогично создаются два дополнительных групповых сообщения. Они используют те же самые теги *Status (Состояние)*, *Lock (Блокировка)* и *Acknowledge (Подтверждение)*, но номера битов соответственно 1 и 2. В правом окне отображаются иконки только что созданных групповых сообщений. 
4	С помощью щелчка R (правой кнопки мыши) на одной из этих иконок можно открыть диалоговое окно *New Single Message(s) (Добавление одиночных сообщений)*. Для каждого группового сообщения вводятся номера соответствующих одиночных сообщений, и диалоговое окно закрывается с помощью кнопки *OK*. 
5	По D (двойному щелчку мыши) на элементе *User–Defined (Определенные пользователем)* в навигационном окне раскрываются конкретные групповые сообщения. Если с помощью (мыши) выбирается один из этих элементов, то в правом окне отобразятся иконки всех добавленных одиночных сообщений. 

Реализация в графическом дизайнере

Шаг	Процедура: Реализация в графическом дизайнере
1	В графическом дизайнере создается новый кадр. В данном случае это кадр *ex_8_generator_00*. В этом кадре конфигурируется *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*, который отображает текущее состояние групповых сообщений. В данном примере это объект *Status Display1*. В соответствии с конфигурацией состояние групповых сообщений сохраняется системой *Alarm Logging* в теге *U16i_ex_alg_20*. Для каждого состояния должен быть разработан соответствующий растровый рисунок. Это означает, что необходимы растровые рисунки для трех неподтвержденных состояний, трех подтвержденных состояний и для состояния готовности. Для атрибута *Properties (Свойства)* → *State (Состояние)* → *Current Status (Текущий статус)* создается *процедура Cи*, которая контролирует состояние в зависимости от тега *U16i_ex_alg_20* и требуемого приоритета. 
2	Дополнительно конфигурируется объект *Windows Object (Объект Windows)* → *Button (Кнопка)*, который меняет текущий кадр на кадр, который инициировал сообщение, если отображается групповое сообщение. В данном случае это объект *Button1*. С помощью *процедуры Cи* для *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* запрашивается текущее состояние группового сообщения и осуществляется смена соответствующего кадра. Если не ожидается никакого сообщения, то дополнительная кнопка помещается поверх только что описанной кнопки и делает ее недоступной. Атрибут *Properties (Свойства)* → *Miscellaneous (Разное)* → *Operator–Control Enable (Разрешение управления оператором)* этой кнопки устанавливается в *No (Нет)*. 
3	Сконфигурируйте другой объект *Windows Object (Объект Windows)* → *Button (Кнопка)*, который используется для подтверждения текущего отображенного группового сообщения. В данном случае это объект *Button3*. С помощью *процедуры Cи* определяется, следует ли подтверждать групповые сообщения и если да, то какие. Если сообщение должно быть подтверждено, соответствующий бит в сконфигурированном теге подтверждения *U16i_ex_alg_22* устанавливается и затем немедленно сбрасывается. Если не ожидается никакого неподтвержденного сообщения, то дополнительная кнопка *Windows Object (Объект Windows)* → *Button (Кнопка)* помещается поверх только что описанной кнопки, чтобы сделать ее недоступной. Атрибут *Properties (Свойства)* → *Miscellaneous (Разное)* → *Operator–Control Enable (Разрешение управления оператором)* этой кнопки устанавливается в *No (Нет)*. 

Шаг	Процедура: Реализация в графическом дизайнере
4	Сконфигурируйте другой кадр, в данном примере это кадр *ex_8_generator_01*. В этом кадре сконфигурированы 3 объекта *Windows Objects (Объекты Windows)* → *Check–Boxes (Флажки)*. В данном случае это объекты *Check–Box1*, *Check–Box2* и *Check–Box3*. Для события *Events (События)* → *Property Topics (Разделы свойств)* → *Output/Input (Вывод/Ввод)* → *Selected Boxes (Выбранные флажки)* создайте *процедуру Cи* для каждого переключателя, который блокирует или делает доступным соответствующее групповое сообщение. Соответствующие блокировки сохраняются системой *Alarm Logging* в теге *U16i_ex_alg_21* согласно конфигурации. Так как блокировка может быть также установлена с другой стороны, то необходимо создать еще одну *процедуру Cи* через для *Properties (Свойства)* → *Output/Input (Ввод/Вывод)* → *Selected Boxes(Выбранные переключатели)*. Эта процедура запускается по изменению тега *U16i_ex_alg_21* и проверяет, изменилось ли состояние блокировки, управляемое соответствующим переключателем. 

Шаг	Процедура: Реализация в графическом дизайнере
5	В первоначально созданном кадре *ex_8_generator_00* создается *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, у которого в поле *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Название кадра)* указывается *ex_8_generator_01*. Установите *Property (Свойство)* → *Miscellaneous (Разное)* → *Display (Отображение)* в *No (Нет)*. Необходимо создать еще один объект *Windows Object* → *Button (Кнопка)*, который делает предварительно сконфигурированное окно кадра видимым с помощью *прямого соединения* для *Events (Событие)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)*. 

Процедура Си для определения текущего состояния

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
WORD state;

state = GetTagWord("U16i_ex_alg_20");

if ((state&1)&&(state&256)) return 6;
else if ((state&2)&&(state&512)) return 5;
else if (state&1) return 3;
else if ((state&4)&&(state&1024)) return 4;
else if (state&2) return 2;
else if (state&4) return 1;
else return 0;
}
```

Чтение тега состояния, записанного системой *Alarm Logging*.

Установка текущего состояния в зависимости от этого тега. Если в состоянии ожидания находится несколько групповых сообщений, то отображение каждого из них определяется приоритетом. В данном случае, приоритет определен следующим образом: запуск начинается с самого высокого уровня приоритета:

Container Alarm (Аварийное сообщение контейнера)

Motor Failure (Сообщение о сбое двигателя)

Acknowledged Container Alarm (Подтверждение аварийного сообщения контейнера)

Container Warning (Предупреждения контейнера)

Acknowledged Motor Failure (Подтверждение сбоя двигателя)

Acknowledged Container Warning (Подтверждение предупреждения контейнера)

Процедура Си для смены кадра

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
Int value;

Value = GetIndex(lpszPictureName,"Status Display1");

If ((value==2)||(value==5))
        SetPictureName("ex_0_startpicture_00.PDL",
        "workspace","ex_3_chapter_02.PDL");
else if (value>0)
        SetPictureName("ex_0_startpicture_00.PDL",
        "workspace","ex_3_chapter_02a.PDL");

}
```

Определение отображенного в настоящий момент на индикаторе состояния.

В зависимости от этого отображенного состояния выполняется смена кадра. Если состояние – 0, никаких действий выполняться не будет.

Процедура Си для подтверждения отображенного сообщения

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{
WORD state;

state = GetTagWord("U16i_ex_alg_20");

if ((state&1)&&(state&256)){
        SetTagWord("U16i_ex_alg_22",
                (WORD)(1|GetTagWord("U16i_ex_alg_22")));
        SetTagWord("U16i_ex_alg_22",
                (WORD)(~1&GetTagWord("U16i_ex_alg_22")));
        }
else if ((state&2)&&(state&512)){
        SetTagWord("U16i_ex_alg_22",
                (WORD)(2|GetTagWord("U16i_ex_alg_22")));
        SetTagWord("U16i_ex_alg_22",
                (WORD)(~2&GetTagWord("U16i_ex_alg_22")));
        }
else if ((state&4)&&(state&1024)){
        SetTagWord("U16i_ex_alg_22",
                (WORD)(4|GetTagWord("U16i_ex_alg_22")));
        SetTagWord("U16i_ex_alg_22",
                (WORD)(~4&GetTagWord("U16i_ex_alg_22")));
        }
}
```

Чтение текущего состояния групповых сообщений.

Если сообщение ожидает подтверждения, оно будет подтверждено. Если подтверждения ожидают несколько сообщений, то будет подтверждено сообщение с самым высоким приоритетом.

Процедура Си для установки блокировки

```
#include "apdefap.h"
void OnPropertyChanged(char* lpszPictureName, char* lpszObjectName, char* l
{
DWORD   dwServiceID;
MSG_RTGROUPSET_STRUCT mGroup;
CMN_ERROR Error;
BOOL fRet;
time_t Time;
struct tm*     TimeStruct;

time(&Time);
TimeStruct = localtime(&Time);

mGroup.stTime.wYear=(WORD)(TimeStruct->tm_year+1900);
mGroup.stTime.wMonth=(WORD)(TimeStruct->tm_mon+1);
mGroup.stTime.wDay=(WORD)(TimeStruct->tm_mday);
mGroup.stTime.wHour=(WORD)(TimeStruct->tm_hour);
mGroup.stTime.wMinute=(WORD)(TimeStruct->tm_min);
mGroup.stTime.wSecond=(WORD)(TimeStruct->tm_sec+1);

mGroup.fIDUsed=FALSE;
strcpy(mGroup.szName,"Alarm Behälter");
mGroup.dwData=value;

MSRTStartMsgService(&dwServiceID,NULL,NULL,
                    MSG_NOTIFY_MASK_ALL,(LPVOID)0,&Error);

fRet=MSRTLockGroup (dwServiceID,&mGroup,&Error);
if (fRet==FALSE)
       printf("Error in MSRTLockGroup(:::) %s\r\n",Error.szErrorText);
else  printf("Executed MSRTLockGroup(:::) \r\n");


MSRTStopMsgService (dwServiceID,&Error );
}
```

Определение требуемых переменных. *mGroup* является структурой, которая должна быть передана функции, отвечающей за установку блокировки.

Определение текущего системного времени. Это значение присваивается элементу структуры *stTime* типа SYSTEMTIME.

Элемент структуры *fIDUsed* указывает, как следует различать групповое сообщение, подлежащее блокированию или разрешению, – по имени или по ID. Значение FALSE определяет, что групповое сообщение идентифицируется по имени.

szName содержит имя группового сообщения.

dwDate указывает, должно ли оно быть заблокировано или разрешено. Текущее состояние передается переключателю.

Запуск службы сообщений с помощью функции *MSRTStartMsgService*.

Вызов функции для блокировки или разрешения группового сообщения *MSRTLockGroup*.

Завершение работы службы сообщений с помощью функции *MSRTStopMsgService*.

Замечание относительно основных применений

В общем случае перед использованием описанных приемов необходимо учесть следующее:

Одиночные сообщения, объединенные в групповое сообщение, должны быть настроены в соответствии с вашими требованиями.

Отображение группового сообщения, приоритет отображения и смена кадров, которые будут выполняться, должны быть настроены в соответствии с вашими требованиями.

4.3 Дизайнер отчетов

В режиме исполнения примеры, имеющие отношение к этой теме, доступны по нажатию (мышью) на *кнопке*, изображенной выше. Примеры приведены в кадре *ex_3_chapter_03.pdl*. В других частях данного демонстрационного проекта также имеются дополнительные примеры.

Общая информация

Дизайнер отчетов (Report Designer) является одной из базовых составных частей WinCC и предоставляет функции для создания и вывода отчетов. Под созданием понимается формирование макета отчета среде конфигурирования *дизайнера отчетов*, а под выводом — печать отчета.

Примечание:

Прилагающиеся системные макеты отчетов можно использовать непосредственно, или можно скопировать и изменить в соответствии с вашими требованиями. Имена системных макетов отчетов и системных заданий печати всегда начинаются с символа @.

4.3.1 Документирование кадра (ex_3_chapter_03.pdl)

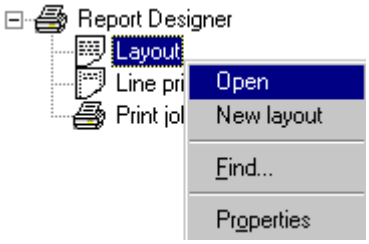
Постановка задачи

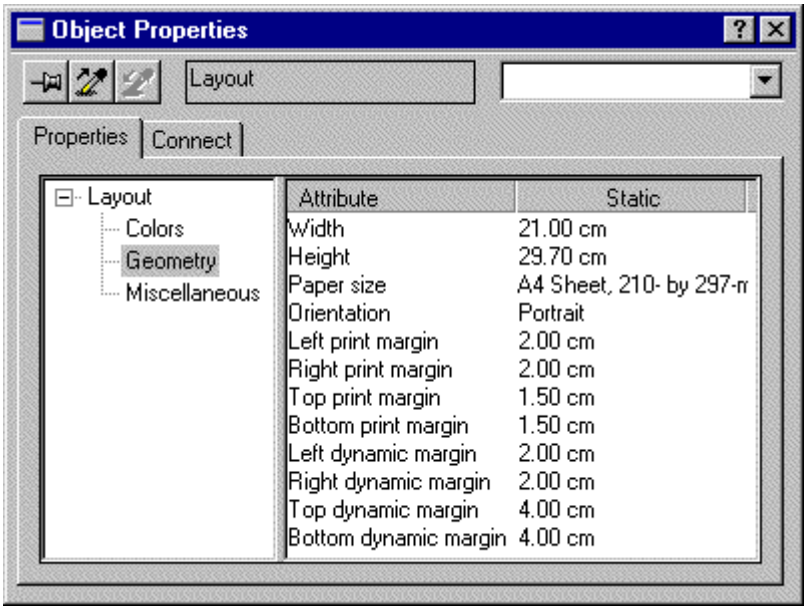
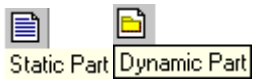
Сформировать исчерпывающую документацию на все кадры, содержащиеся в проекте. Для каждого кадра описание должно включать графическое изображение кадра, общую информацию о кадре, список всех объектов и список всех заданных атрибутов кадра.

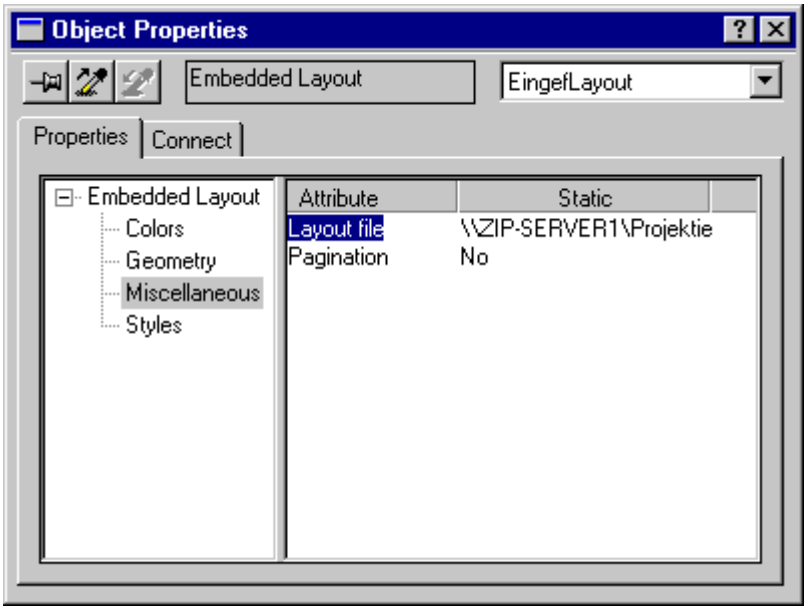
Концепция реализации

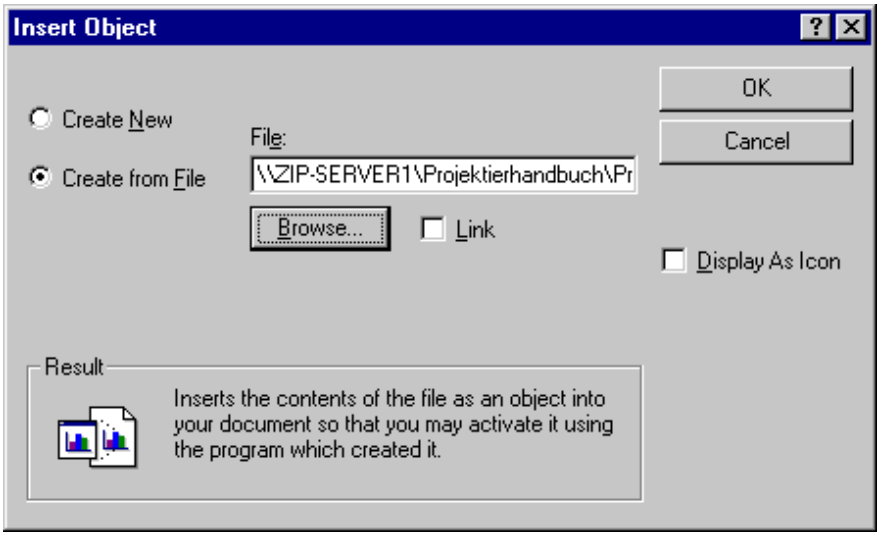
Имеется системный макет отчета, отвечающий предъявленным требованиям. Это макет *@PDL picture (compact).rpl*. Этот макет можно скопировать и внести в него необходимые вам изменения.

Реализация в дизайнере отчетов

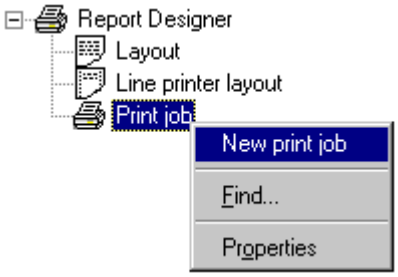
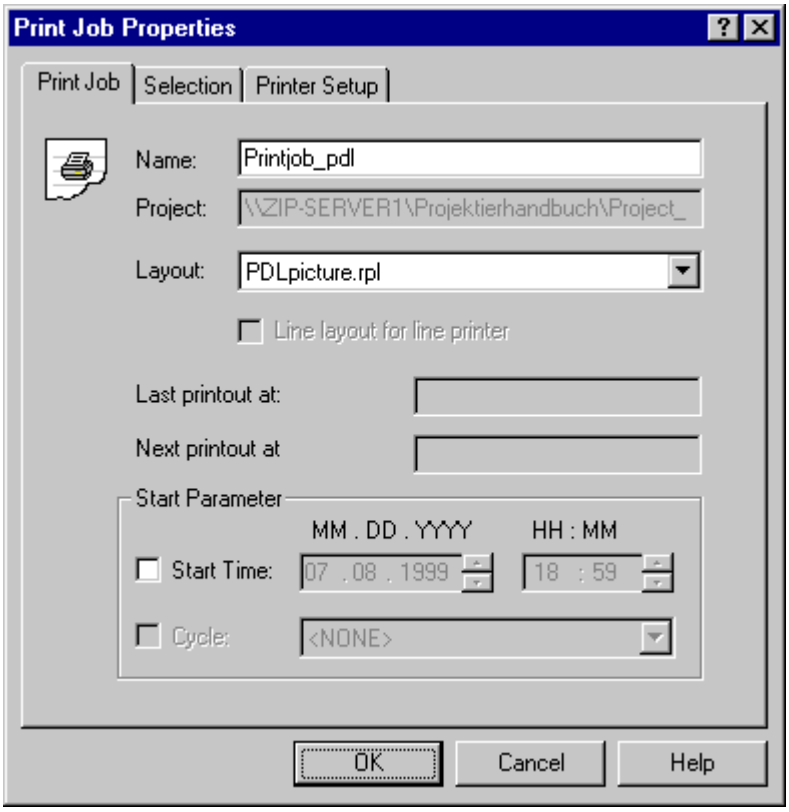
Шаг	Процедура: Реализация в дизайнере отчетов
1	Откройте *дизайнер отчетов* из *проводника WinCC*. 
2	При помощи команды меню *File (Файл)* → *Open...(Открыть…)* откройте системный макет отчета *@PDLPic.rpl* и сохраните его под другим именем командой *File (Файл)* → *Save As... (Сохранить как…)*. В данном примере используется имя *PDLpicture.rpl*.

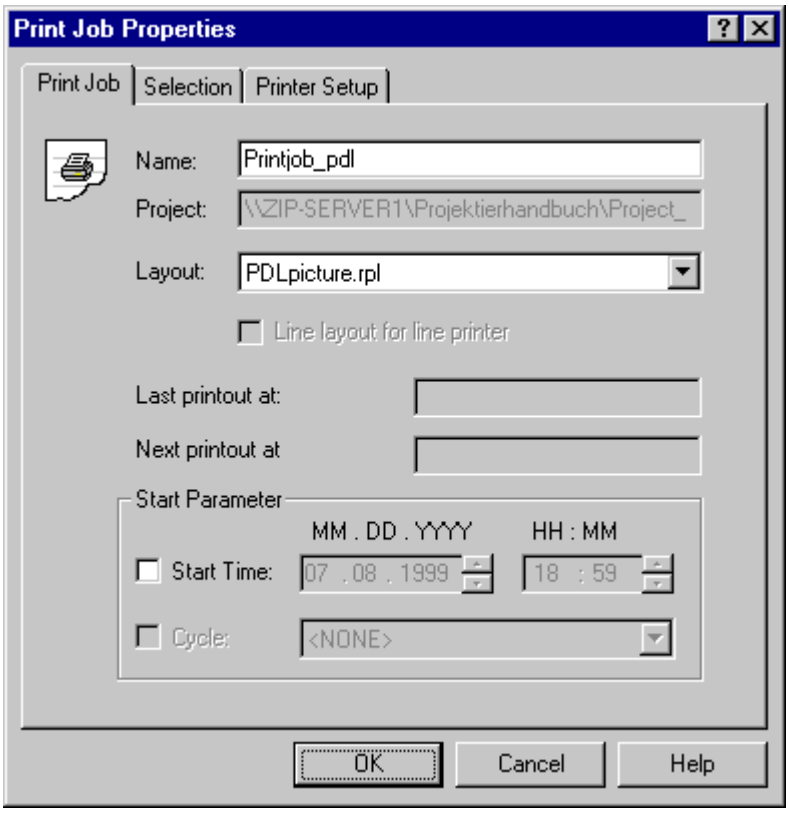
Шаг	Процедура: Реализация в дизайнере отчетов
3	По щелчку R (правой кнопки мыши) на пустом месте макета отчета откройте диалоговое окно свойств. На закладке *Properties (Свойства)* в разделе *Geometry (Геометрия)* можно указать основные геометрические параметры. В разделе *Miscellaneous (Разное)* вы можете описать титульную и последнюю страницы отчета. В данном примере настраивается титульная страница. 
4	Используя изображенные ниже кнопки панели инструментов можно редактировать статическую и динамическую части отчета. 

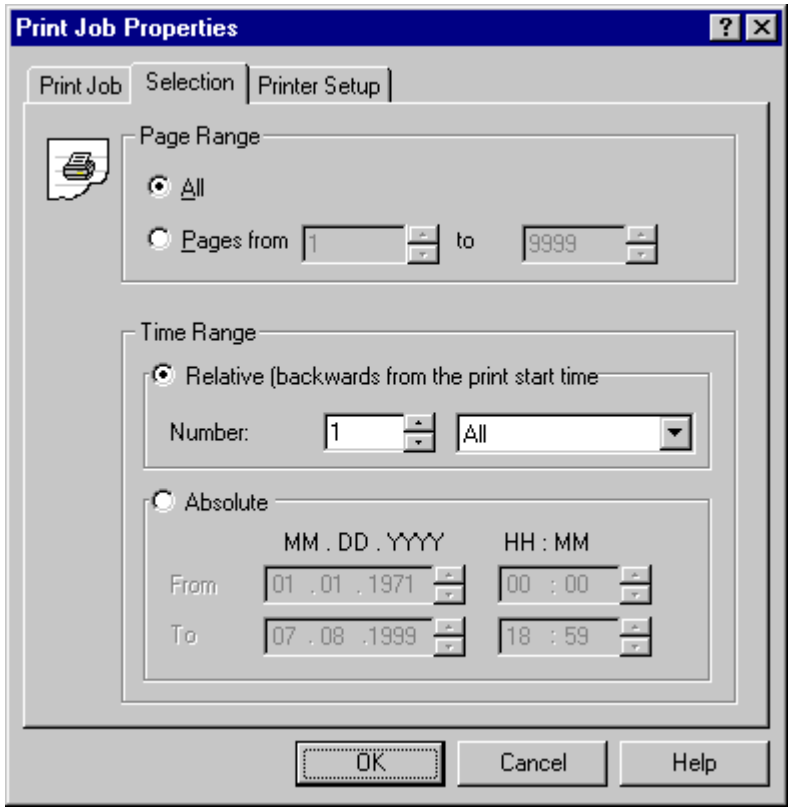
Шаг	Процедура: Реализация в дизайнере отчетов
5	Динамическая часть отчета содержит *Dynamic Object (Динамический Объект)* → *Embedded Layout (Встроенный макет)*. В данном примере это объект *EmbedLayout*, который нужно изменить. По щелчку R (правой кнопки мыши) на динамической части отчета выберите *@PDL picture (compact).rpl* в пункте *Properties (Свойства)* → *Miscellaneous (Разное)* → *Layout File (Файл макета)*. Этот макет можно открыть и изменить его элементы в соответствии с вашими требованиями. Однако рекомендуется сначала скопировать макет, и затем уже модифицировать копию. При этом вновь созданный макет следует указать в качестве макета для *EmbedLayout* в пункте *Properties (Свойства)* → *Miscellaneous (Разное)* → *Layout File (Файл макета)*. 

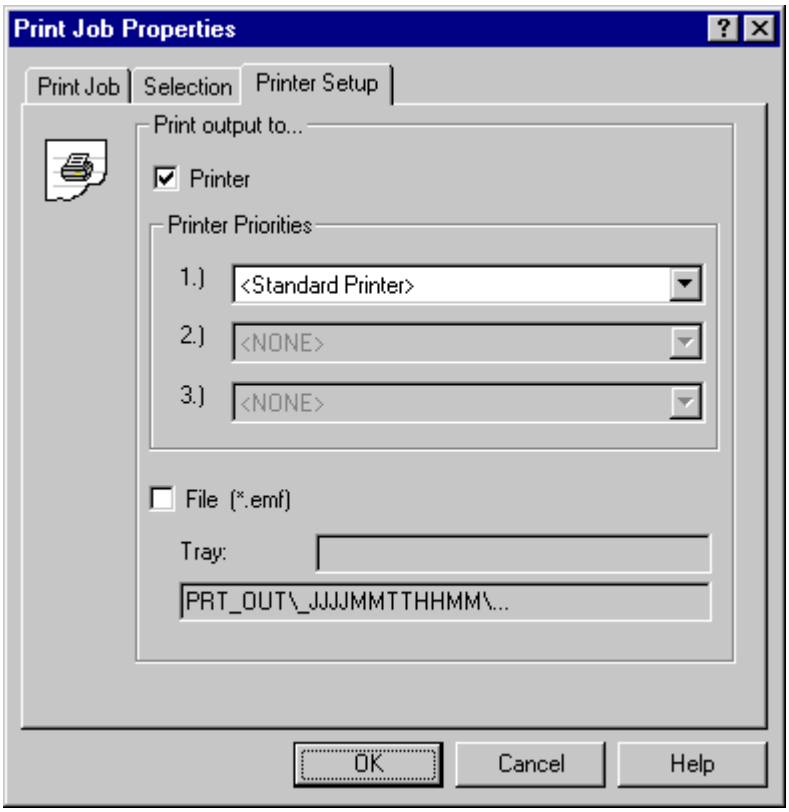
Шаг	Процедура: Реализация в дизайнере отчетов
6	Статическая часть отчета содержит верхний и нижний колонтитулы. В нижнем колонтитуле находятся *системные объекты Date/Time (Дата/Время)*, *Page Number (Номер страницы)*, *Project Name (Имя проекта)* и *Layout Name (Имя макета)*. Нижний колонтитул содержит два объекта *Static Objects (Статический объект)* → *Static Texts (Статический текст)*, а также *System Object (Системный объект)* → *Project Name (Имя проекта)*. Кроме того, при помощи *Static Object (Статический объект)* → *OLE Element (Элемент OLE)* отображается логотип. В данном примере текст объекта *StatText1* сменен на *Picture Documentation*. Для отображения вашего собственного логотипа удалите существующий объект *OLEElement1*. Создайте новый *Static Object (Статический объект)* → *OLE Element (Элемент OLE)*. В диалоге *Insert Object (Вставка объекта)*, который появляется после добавления объекта в отчет, выберите опцию *Create from File (Создать из файла)* и укажите файл кадра, который содержит логотип. Диалог завершается нажатием на *OK*. 
7	При помощи кнопок панели инструментов, изображенных ниже, вы можете переключаться между титульной страницей, телом отчета и последней страницей. В данном примере титульный лист содержит два объекта *Static Objects (Статический объект)* → *Static Texts (Статический текст)*, *System Object (Системный объект)* → *Project Name (Имя проекта)* и *Static Object (Статический объект)* → *Static Metafile (Статический метафайл)*. 
8	Изменения, сделанные в дизайнере отчетов, сохраняются, приложение закрывается.

Создание задания печати (Print Job)

Шаг	Процедура: Создание задания печати
1	В *проводнике WinCC* новое задание печати создается щелчком R (правой кнопки мыши). 
2	Это новое задание с именем *Print Job001* будет добавлено к существующим заданиям в правом окне. По D (двойному щелчку мыши) или R (щелчку правой кнопки) на этом задании откроется диалоговое окно свойств. 

Шаг	Процедура: Создание задания печати
3	На закладке *Print Job (Задание печати)* имя, использующееся по умолчанию, меняется на *Printjob_pdl*. В поле *Layout (Макет)* указывается ранее созданный макет *PDLpicture.rpl*. 

Шаг	Процедура: Создание задания печати
4	На закладке *Selection (Выбор)* вы можете определить, что необходимо печатать. В поле *Page Range (Диапазон страниц)* выбрана опция *All (Все)*. Параметр *Time Range (Диапазон времени)* не будет иметь значения в данном примере. 

Шаг	Процедура: Создание задания печати
5	Используемый принтер указывается на закладке *Printer Setup (Настройка Принтера)*. Данные могут быть также выведены в файл. 
6	Щелчком R (правой кнопки мыши) задание печати может быть запущено из *проводника WinCC*. Таким же образом можно выполнить предварительный просмотр. 
7	В проекте–примере предварительный просмотр задания печати можно активизировать при помощи объекта *Windows Object (Объекты Windows)* → *Button (Кнопка)*. Это объект *Button13* в кадре *ex_3_chapter_03.pdl*. Для события *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* создайте *процедуру Cи*, которая запускает предварительный просмотр задания печати.

Процедура Си для запуска задания печати

```
#include "apdefap.h"
void OnClick(char* lpszPictureName, char* lpszObjectName, char* lpszPropert
{

RPTJobPreview("PrintJob_pdl");

}
```

Вызывается стандартная функция *RPTJobPreview*. В качестве параметра передается имя задания печати.

Примечание:

Если макет, внедренный в отчет, открыт в *дизайнере отчетов*, то предварительный просмотр или задание печати не могут быть выполнены.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

Созданный макет отчета может быть использован без изменений. Логотип и информацию, включаемую в отчет, следует изменить в соответствии с вашими требованиями. Эти изменения должны быть выполнены согласно указаниям в пункте *Реализация в дизайнере отчетов*, шаги 3 и 4.

4.3.2 Отчет проводника WinCC (ex_3_chapter_03.pdl)

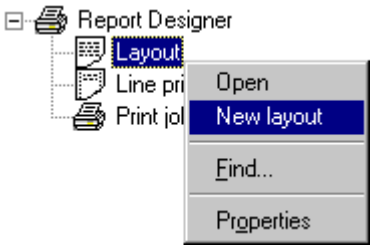
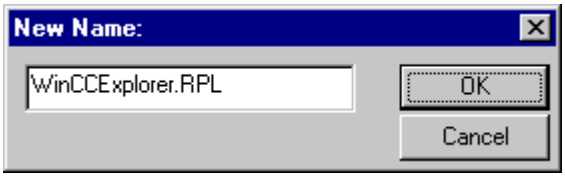
Постановка задачи

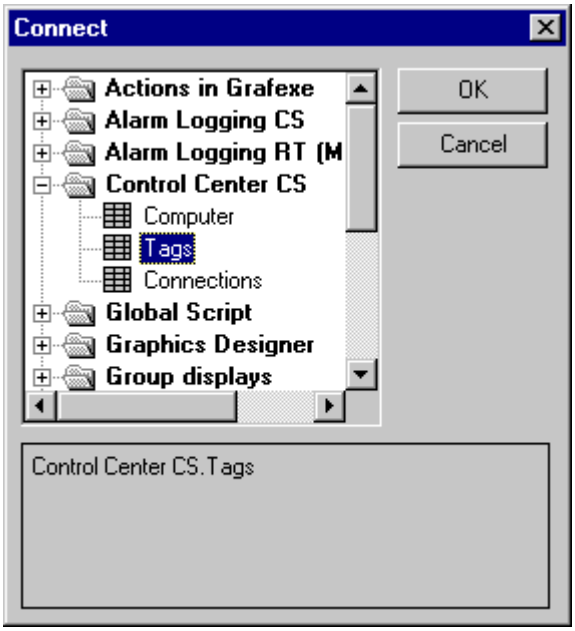
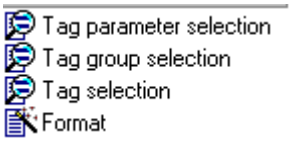
Сформировать описание всех тегов, входящих в определенные группы, используемые в проекте. Описание должно включать имя тега, тип тега, группу тегов, параметры тегов и информацию, относящуюся к подключению.

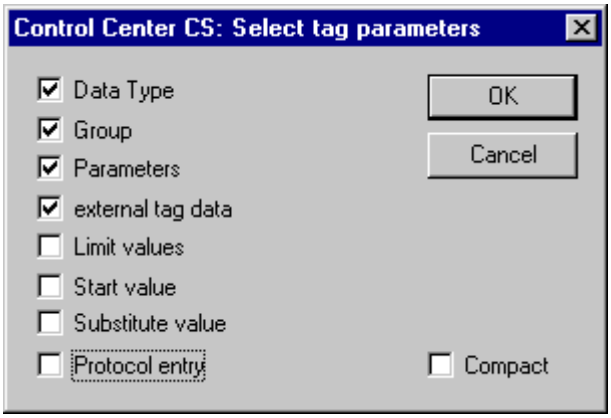
Концепция реализации

В *дизайнере отчетов* создается специальный макет. Этот макет не базируется ни на одном готовом макете.

Реализация в дизайнере отчетов

Шаг	Процедура: Реализация в дизайнере отчетов
1	Щелчком R (правой кнопки мыши) в разделе *Page Layout (Макет страницы) проводника WinCC* создается новый макет отчета. 
2	Этот новый макет, называющийся по умолчанию *NewRPL00.RPL*, будет добавлен в правом окне к существующим макетам страниц. Этот макет можно переименовать по щелчку R (правой кнопки мыши). В данном примере используется имя *WinCC Explorer.rpl*. 
3	В *дизайнере отчетов* откройте новый макет. В диалоговом окне свойств макета, доступном по щелчку R (правой кнопки мыши) на пустом месте страницы отчета, в разделе *Miscellaneous (Разное)* указывается *Cover Sheet (Титульный лист)*. Остальные значения не изменяются.
4	В статической части отчета для верхнего и нижнего колонтитулов конфигурируются различные *Static Objects (Статические объекты)* и *System Objects (Системные объекты)*. Внешний вид титульной страницы является лишь примером для вашего собственного дизайна.

Шаг	Процедура: Реализация в дайнере отчетов
5	В динамической части отчета настраивается *Dynamic Object (Динамический объект)* → *Dynamic Table (Динамическая таблица)*. В данном примере используется объект *DynTable1*. После добавления объекта в отчет появляется диалоговое окно *Connect (Соединение)*. В папке *проводника WinCC* выберите элемент *Tag (Тег)*. Закройте диалоговое окно нажатием на *OK*. 
6	На закладке *Connect (Соединение)* диалога свойств таблицы приводятся некоторые опции. 
7	По D (двойному щелчку мыши) на этих пунктах вызывается диалоговое окно для выбора данных. Наличие выбранных данных обозначается красной меткой .

Шаг	Процедура: Реализация в дизайнере отчетов
8	В диалоговом окне выбора параметров тегов включаются флажки *Data type (Тип данных)*, *Group (Группа)*, *Parameters (Параметры)* и *External Tag Data (Данные внешних тегов)*. Дополнительно включается флажок *Compact (Сжатие)*. В результате этого все данные тегов отображаются в одной строке. 
9	В качестве выбранных групп тегов в данном примере используются *AlarmLogging1* и *AlarmLogging2*. Такой выбор возможен только при снятом флажке *All Tag Groups (Все группы тегов)*. Для выбора отдельных групп тегов флажок *All Tags (Все теги)* в диалоговом окне выбора тегов должен быть сброшен. Все изменения, сделанные в *дизайнере отчетов*, необходимо сохранить.

Примечание:

Процедура создания нового задания печати и его запуска из *проводника WinCC* и режима исполнения описана в первом примере главы *Report Designer (Дизайнер отчетов)* в разделе Создание заданий печати. Настройка производится аналогичным образом.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

Созданный макет отчета можно использовать без каких–либо изменений.

4.3.3 Отчет системы конфигурирования регистрации тегов (ex_3_chapter_03.pdl)

Постановка задачи

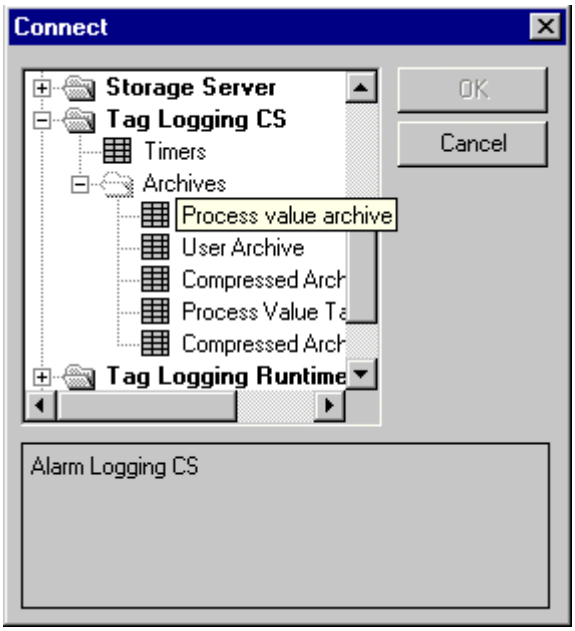
Необходимо сформировать документацию на все использующиеся в проекте архивы параметров процесса. Описание должно включать как общие параметры архива, так и настройки отдельных архивных тегов.

Концепция реализации

В *дизайнере отчетов* создается специальный макет. Этот макет не базируется ни на одном готовом макете.

Реализация в дизайнере отчетов

Шаг	Процедура: Реализация в дизайнере отчетов
1	Щелчком R (правой кнопки мыши) в разделе *Page Layout (Макет страницы) проводника WinCC* создается новый макет отчета. Этот новый макет, называющийся по умолчанию *NewRPL00.RPL*, будет добавлен в правом окне к существующим макетам страниц. По щелчку R (правой кнопки мыши) на этом имени переименуйте макет в *tlg_cs.rpl*.
2	В *дизайнере отчетов* откройте новый макет. В диалоговом окне свойств макета, доступном по щелчку R (правой кнопки мыши) на пустом месте страницы отчета, в разделе *Miscellaneous (Разное)* указывается *Cover Sheet (Титульный лист)*. Остальные значения не изменяются.
3	В статической части отчета для верхнего и нижнего колонтитулов конфигурируются различные *Static Objects (Статические объекты)* и *System Objects (Системные объекты)*. Внешний вид титульной страницы является лишь примером для вашего собственного дизайна.

Шаг	Процедура: Реализация в дизайнере отчетов
4	В динамической части отчета настраивается *Dynamic Object (Динамический объект)* → *Dynamic Table (Динамическая таблица)*. В данном примере используется объект *DynTable1*. После добавления объекта в отчет появляется диалоговое окно *Connect (Соединение)*. В папке *Tag Logging CS (Система конфигурирования регистрации тегов)* выберите пункт *Process Value Archive (Архив параметров процесса)*. Закройте диалоговое окно нажатием на *OK*. 
5	На закладке *Connect (Соединение)* диалога свойств таблицы приводятся некоторые опции. Archive names Process archive data
6	По D (двойному щелчку мыши) на этих пунктах вызывается диалоговое окно для выбора данных. Наличие выбранных данных обозначается красной меткой . В диалоговом окне выбора архивов включен переключатель *All Archives (Все архивы)*. В диалоговом окне выбора архивных данных помечены все имеющиеся пункты. Все изменения, сделанные в *дизайнере отчетов*, нужно сохранить.

Примечание:

Процедура создания нового задания печати и его запуска из *проводника WinCC* и режима исполнения описана в первом примере главы *Дизайнер отчетов* в разделе Создание заданий печати. Настройка производится аналогичным образом.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

Созданный макет отчета можно использовать без каких–либо изменений.

4.3.4 Печать окна трендов в режиме исполнения (ex_3_chapter_01a.pdl)

Постановка задачи

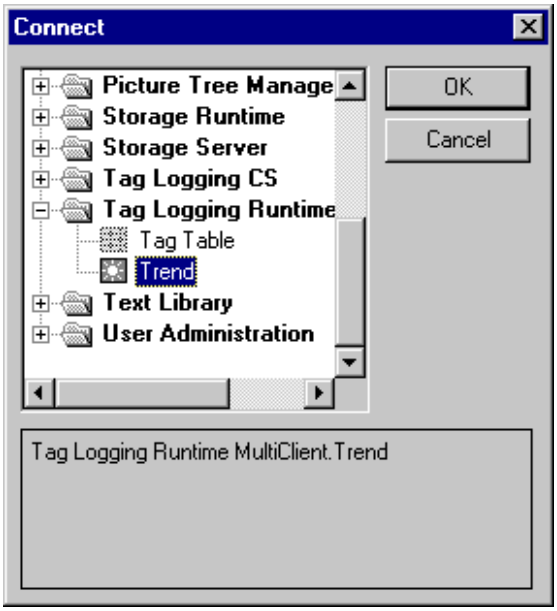
В режиме исполнения вывести на печать содержимое окна трендов. Необходимо иметь возможность выбирать диапазон времени для распечатываемых данных. Этот пример базируется на примере Cyclic–Selective Archiving (Выборочная циклическая архивация) (ex_3_chapter_01a.pdl) в разделе *Tag Logging (Регистрация тегов)*. Он используется для вывода на печать отображаемой в данном примере таблицы.

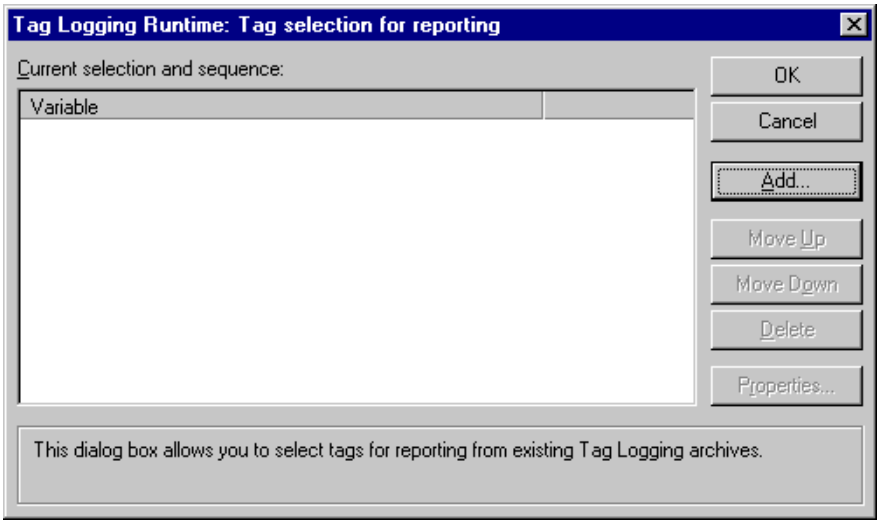
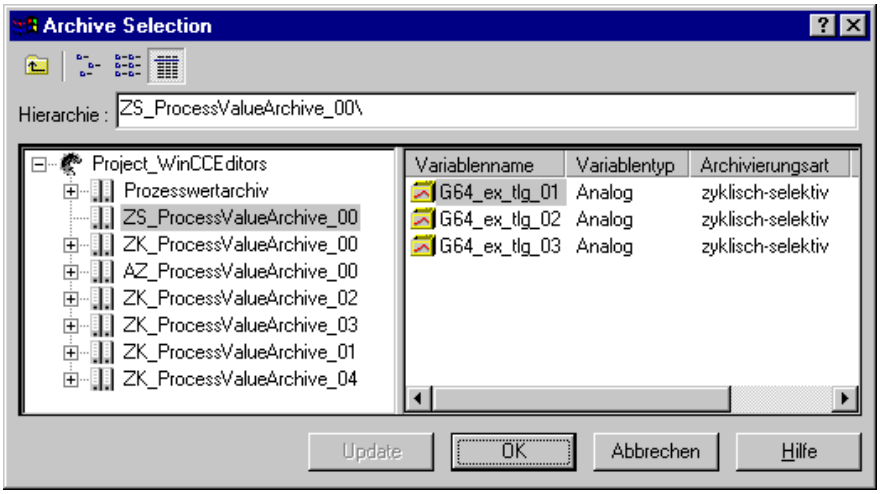
Концепция реализации

В *дизайнере отчетов* создается специальный макет. Выбор временного диапазона производится не на макете, а в режиме исполнения с помощью функции проекта. Эта функция будет выбирать диапазон времени непосредственно у задания печати.

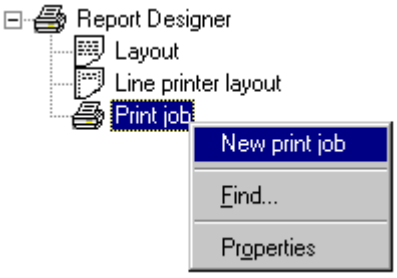
Реализация в дизайнере отчетов

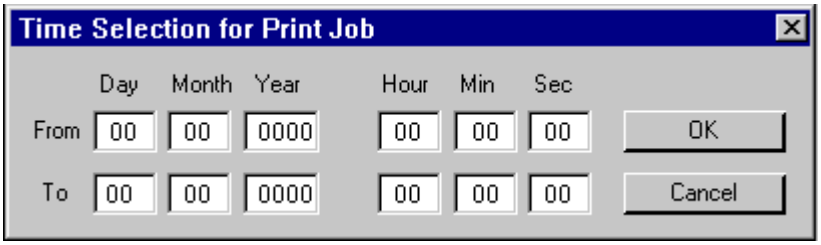
Шаг	Процедура: Реализация в дизайнере отчетов
1	Щелчком  R (правой кнопки мыши) в разделе *Page Layout (Макет страницы) проводника WinCC* создается новый макет отчета. Этот новый макет будет добавлен в правом окне к уже существующим макетам страниц. Имя, данное макету по умолчанию, следует изменить щелчком  R (правой кнопки мыши) на *tlg_ZS_PA_00.rpl*.
2	В *дизайнере отчетов* откройте новый макет. В статической части отчета для верхнего и нижнего колонтитулов конфигурируются различные *Static Objects (Статические объекты)* и *System Objects (Системные объекты)*.

Шаг	Процедура: Реализация в дизайнере отчетов
3	В динамической части отчета сконфигурируйте три объекта *Dynamic Objects (Динамический объект)* → *Dynamic Metafiles (Динамический метафайл)*. В данном примере используются объекты *DynMetafile1*, *DynMetafile2* и *DynMetafile3*. После добавления объектов в отчет появится диалоговое окно *Connect (Соединение)*. Для всех трех объектов выберите пункт *Tag Trend (Тренд тега)* из папки *Tag Logging Runtime (Система исполнения регистрации тегов)*. Диалоговое окно закрывается нажатием на *OK*. 

Шаг	Процедура: Реализация в дизайнере отчетов
4	На закладке *Connect (Соединение)* диалогового окна *Dynamic Metafile's (Динамические метафайлы)* приводятся некоторые опции. Time range Tag Selection Format По D (двойному щелчку мыши) на этих пунктах вызывается соответствующее диалоговое окно для выбора данных. Наличие выбранных данных обозначается красной меткой . *Time Selection (Выбор времени)* не производится. В окне *выбора тегов* диалог *Tag Selection for Reporting (Выбор тегов для отчета)* вызывается щелчком (мыши) → *Edit (Правка)*. 
5	Сделанные изменения в *дизайнере отчетов* следует сохранить. По щелчку (мыши) → *Add (Добавить)* открывается диалоговое окно *Archive Data Selection (Выбор архивных данных)*. Выбираются архивы *ZS_ProcessValueArchive_00* и tag G64_ex_tlg_01. Диалог можно закрыть по нажатию на *OK*. Таким же образом настройте оставшиеся два тега для других объектов. 

Создание задания печати (Print Job)

Шаг	Процедура: Создание задания печати
1	В *проводнике WinCC* новое задание печати создается щелчком R (правой кнопки мыши) на пункте *Print Job (Задание печати)*.  Это задание, названное по умолчанию *Print Job001*, будет добавлено в правом окне к уже существующим заданиям печати. Диалоговое окно свойств открывается по D (двойному щелчку мыши) или щелчку R (правой кнопки мыши) на этом задании. На закладке *Print Job (Задание печати)* введено *Name (Имя) Printjob_ZS_PA_00*. В качестве *макета* указывается ранее созданный файл *tlg_ZS_PA_00.rpl*. На закладке *Selection (Выбор)* установлен диапазон печати. В поле *Page Range (Диапазон страниц)* выбрана опция *All (Все)*. В поле *Time Range (Диапазон времени)* выбрана опция *Absolute (Абсолютный)*. Временной диапазон не задан, его укажем позже в режиме исполнения. На закладке *Printer Setup (Настройка принтера)* выбирается используемый для печати принтер.
2	В примере проекта при помощи объекта *Windows Object (Объект Windows)* → *Button (Кнопка)* можно выполнить предварительный просмотр задания печати. Это объект *Button13* в кадре *ex_3_chapter_01a.pdl*. Для события *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* создайте *процедуру Cu*, которая будет запускать предварительный просмотр задания печати. 

Шаг	Процедура: Создание задания печати
3	Для выбора временного интервала необходимо диалоговое окно. В качестве диалога используется отдельный кадр, в данном примере это *ex_5_window_02.PDL*. В этом кадре размещено шесть пар объектов *Smart Objects (Интеллектуальные объекты)* → *I/O Fields (Поля ввода/вывода)*, каждая пара служит для ввода начала и конца временного диапазона. Начальные значения интервалов задаются объектами с I/O Field1 по I/O Field6, конечные — объектами с I/O Field7 по I/O Field12. Для буферизации вводимых значений в менеджере тегов для каждого объекта *I/O Field (Поле ввода/вывода)* создается тег типа *Unsigned 16–Bit Value (16–битная величина без знака)*. В данном примере это теги с *U16i_ex_rep_f1* по *U16i_ex_rep_f6* для начал интервалов, и с *U16i_ex_rep_t1* по *U16i_ex_rep_t6* для концов интервалов. Для каждого объекта *I/O Field (Поле ввода/вывода)* в поле *Properties (Свойства)* → *Output/Input (Вывод/ввод)* → *Output Value (Выходное значение)* создается *соединение с тегом*. В атрибутах *Properties (Свойства)* → *Output/Input (Вывод/ввод)* → *Output Format (Формат вывода)* укажите формат вида *099* для всех объектов *I/O Fields (Поля ввода/вывода)*, кроме тех, что содержат значение года. Для таких полей формат *09999*. 
4	Для события *Events (События)* → *Miscellaneous (Разное)* → *Open Picture (Открытие кадра)* кадра *ex_5_window_02.PDL* создается *процедура Cu*, которая записывает в теги предопределенные временные значения. В качестве конца диапазона времени используется текущее системное время, а в качестве начала — текущее системное время минус одна минута.
5	В кадре *ex_5_window_02.PDL* размещаются два объекта типа *Windows Objects (Объекта Windows)* → *Buttons (Кнопки)*. В данном примере используются объекты *Button1* и *Button2*. *Button2* — кнопка отмены. Для события *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* создается *прямое соединение*, которое у *Current Window (Текущее окно)* переключает признак *Display (Отображение)* в *константу 0*. *Button1* — кнопка OK. Для закрытия окна у нее также создается *прямое соединение* для события *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)*. Для события *Events (События)* → *Mouse (Мышь)* → *Press left (Нажатие левой кнопки)* создается *процедура Cu*. Эта процедура вызывает созданную ранее функцию проекта для выбора временного интервала у задания печати. Имя задания печати считывается из тега типа *Text Tag 16–Bit Character Set (Текстовый тег 16–битного набора символов)*, созданного в *менеджере тегов*. В данном примере это тег *T16i_ex_rep_00*.

Шаг	Процедура: Создание задания печати
6	Для отображения созданного кадра в *ex_3_chapter_01a.PDL* нужно добавить *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. В данном примере используется объект *Picture Window1*. В атрибуте *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* указывается ранее созданный кадр *ex_5_window_02.PDL*. Установите свойство *Display (Отображать)* в *No*.
7	Для того чтобы сделать *окно кадра* видимым, необходим дополнительный объект *Windows Object (Объект Windows)* → *Button (Кнопка)*, в данном примере это кнопка *Button12*. У этого объекта создается *процедура Cи* для события *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)*, которая записывает имя задания печати в тег *T16i_ex_rep_00* и делает объект *Picture Window1* видимым. 

Процедура Cи для кнопки OK

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszF
{

ModifyPrintJob(TimeFrom(),
        TimeTo(),
        GetTagChar("T16i_ex_rep_00"));

}
```

Вызывается функция проекта *ModifyPrintJob*. Этой функции в качестве параметров требуется два значения времени в формате структуры *SYSTEMTIME*. Эти значения считываются из тегов, хранящих границы временного диапазона, при помощи двух функций проекта *TimeFrom()* и *TimeTo()*. Дополнительно необходимо задать имя задания печати. Это имя хранится в теге *T16i_ex_rep_00*.

Функция проекта ModifyPrintJob

```
BOOL ModifyPrintJob(SYSTEMTIME st1,SYSTEMTIME st2,char jobname[200])
{
BOOL            fRet;
PCMN_ERROR      pError;
HPROPERTIES     hProp;
LPVOID  ptr1,ptr2;
DWORD   typ;
DWORD   dwVal;
char            propname1[200],propname2[200];
TCHAR           g_szProj[MAX_PATH+1];

typ = VT_DATE;
strcpy( propname1, "ABSOLUTESELECTIONFROM");
strcpy( propname2, "ABSOLUTESELECTIONTO");
ptr1 = (LPVOID)&st1;
ptr2 = (LPVOID)&st2;

//-----get project path
if( !DMGetRuntimeProject( g_szProj, MAX_PATH, pError)) {
        printf("Error DMGetRuntimeProject(...)\r\n");
        return FALSE;
        }
//-----create property handle
hProp = RPJCreatePropertyHandle ( g_szProj, pError );
if( !hProp) {
        printf("Error RPJCreatePropertyHandle(...)\r\n");
        return FALSE;
        }
//-----get job properties
if ( !RPJGetJobProps ( hProp, jobname, pError )) {
        printf("Error RPJGetProps(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
        }
//-----set property
if ( !RPJSetProperty ( hProp, propname1, ptr1,
                (VARTYPE) typ, 200, pError )) {
        printf("Error RPJSetProperty(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
        }
//-----save job properties
if ( !RPJSetJobProps ( hProp, jobname, pError)) {
        printf("Error RPJSetProps(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
        }
//-----get job properties
if ( !RPJGetJobProps ( hProp, jobname, pError )) {
        printf("Error RPJGetProps(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
        }
//-----set property
if ( !RPJSetProperty ( hProp, propname2, ptr2,
                (VARTYPE) typ, 200, pError )) {
        printf("Error RPJSetProperty(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
        }
//-----save job properties
if ( !RPJSetJobProps ( hProp, jobname, pError)) {
        printf("Error RPJSetProps(...)\r\n");
        RPJDeletePropertyHandle ( hProp, pError);
        return FALSE;
        }
//-----delete property handle
fRet = RPJDeletePropertyHandle ( hProp, pError);
return TRUE;
}
```

В качестве передаваемых в формате структуры *SYSTEMTIME* параметров *st1* и *st2* функция получает значения начала и конца временного интервала, которые необходимо установить.

С помощью функции *DMGetRuntimeProject* определяется путь к проекту.

Устанавливается и сохраняется время начала. Это атрибут *ABSOLUESELECTIONFROM*.

Устанавливается и сохраняется время конца. Это атрибут *ABSOLUESELECTIONTO*.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

В макете, используемом для печати трендов, необходимо указать включаемые в отчет архивы и распечатываемые архивные теги.

Диалоговое окно для выбора времени можно оставить без изменений. Для его функционирования требуются функции проекта *ModifyPrintJob*, *TimeFrom* и *TimeTo*. Теги для сохранения значений времени должны быть созданы с такими же именами. В противном случае следует изменить функции *TimeFrom* и *TimeTo*. При использовании диалога для нескольких заданий печати рекомендуется создать текстовый тег для хранения их имен.

4.3.5 Печать таблиц в режиме исполнения (ex_3_chapter_01c.pdl)

Постановка задачи

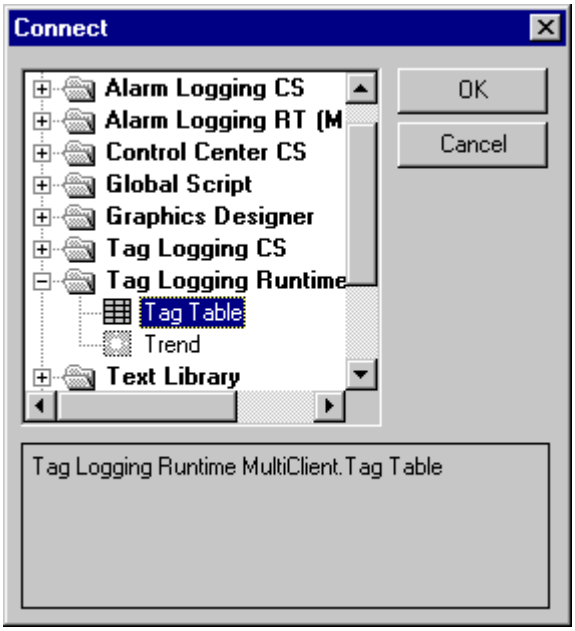
Напечатать таблицу в режиме исполнения. Необходимо иметь возможность выбора диапазона времени для распечатываемых данных.
Этот пример основан на примере User–Defined Table Layout (Определяемый пользователем макет таблицы) (ex_3_chapter_01c.pdl) в разделе *Tag Logging (Регистрация тегов)*. Он используется для вывода на печать отображаемой в данном примере таблицы.

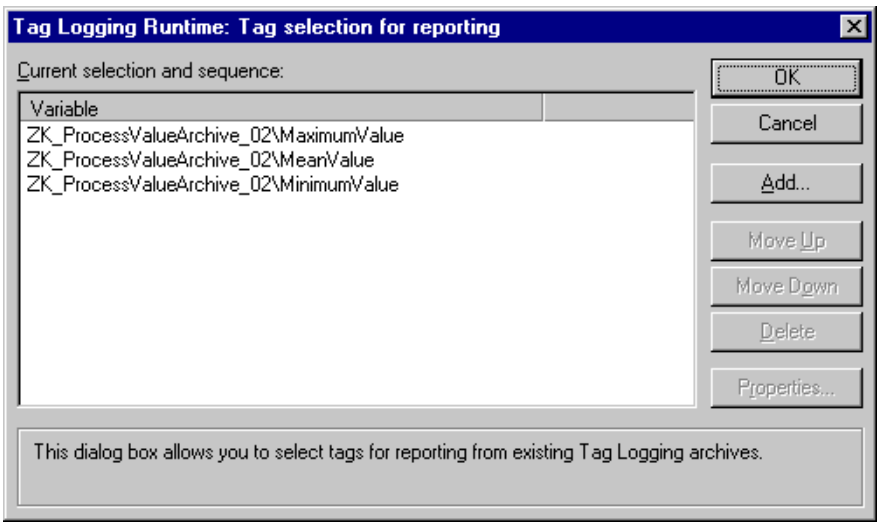
Концепция реализации

В *дизайнере отчетов* создается специальный макет. Выбор временного диапазона производится не на макете, а в режиме исполнения с помощью функции проекта. Эта функция будет выбирать диапазон времени непосредственно у задания печати.
Процедура для выбора временного интервала в режиме исполнения описана в предыдущем примере *Creation of a Print Job (Создание задания печати)*.

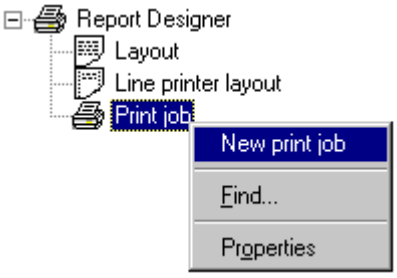
Реализация в дизайнере отчетов

Шаг	Процедура: Реализация в дизайнере отчетов
1	Щелчком R (правой кнопки мыши) в разделе *Page Layout (Макет страницы) проводника WinCC* создается новый макет отчета. Этот новый макет будет добавлен в правом окне к уже существующим макетам страниц. Имя, данное макету по умолчанию, следует изменить щелчком R (правой кнопки мыши) на *tlg_ZS_PA_00.rpl*.
2	В *дизайнере отчетов* откройте новый макет. В статической части отчета для верхнего и нижнего колонтитулов конфигурируются различные *Static Objects (Статические объекты)* и *System Objects (Системные объекты)*.

Шаг	Процедура: Реализация в дизайнере отчетов
3	В динамической части отчета поместите объект *Dynamic Object (Динамический объект)* → *Dynamic Table (Динамическая таблица)*. В данном примере это объект *DynTable1*. После добавления объекта в отчет появится диалоговое окно *Connect (Соединение)*. В папке *Tag Logging Runtime (Система исполнения регистрации тегов)* выберите *Tag Table (Таблица тегов)*. Диалоговое окно закрывается нажатием на *OK*. 

Шаг	Процедура: Реализация в дайнере отчетов
4	На закладке *Connect (Соединение)* диалогового окна *Dynamic Metafile (Динамический метафайл)* приводятся некоторые опции. Time range Tag Selection По D (двойному щелчку мыши) на этих пунктах вызывается соответствующее диалоговое окно для выбора данных. Наличие выбранных данных обозначается красной меткой . *Time Selection (Выбор времени)* не производится. По щелчку (мыши) → *Edit (Правка)* → *Add (Добавить)* вызывается диалоговое окно для выбора архивных значений. В этом окне выбирается архив *ZK_ProcessValueArchive_00* и содержащиеся в нем архивируемые теги. Диалоговое окно закрывается нажатием на OK.  Изменения, сделанные в *дайнере отчетов*, нужно сохранить.

Создание задания печати

Шаг	Процедура: Создание задания печати
1	В *проводнике WinCC* новое задание печати создается щелчком R (правой кнопки мыши).  Это задание, названное по умолчанию *Print Job001*, будет добавлено в правом окне к уже существующим заданиям печати. Диалоговое окно свойств открывается по D (двойному щелчку мыши) или щелчку R (правой кнопки мыши) на этом задании. На закладке *Print Job (Задание печати)* введено *Name (Имя) Printjob_ZS_PA_02*. В качестве *макета* указывается ранее созданный файл *tlg_ZS_PA_02.rpl*. На закладке *Selection (Выбор)* установлен диапазон печати. В поле *Page Range (Диапазон страниц)* выбрана опция *All (Все)*. В поле *Time Range (Диапазон времени)* выбрана опция *Absolute (Абсолютный)*. Временной диапазон не задан, его укажем позже в режиме исполнения. На закладке *Printer Setup (Настройка принтера)* выбирается используемый для печати принтер.
2	Процедура для выбора временного интервала в режиме исполнения описана в предыдущем примере *Создание задания печати*. У кнопки, изображенной ниже, необходимо изменить *процедуру Си*, связанную с событием *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)*.  В текстовый тег *T16i_ex_rep_00* необходимо записать название только что созданного задания печати. У кнопки, запускающей предварительный просмотр, также необходимо изменить название задания печати в вызове *процедуры Си* для события *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)*. 

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

Разработанный макет может быть использован сразу после выбора архива.

4.3.6 Отчет последовательности сообщений (ex_3_chapter_02b.pdl)

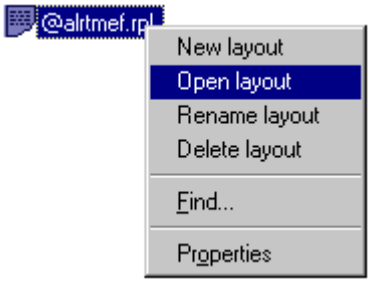
Постановка задачи

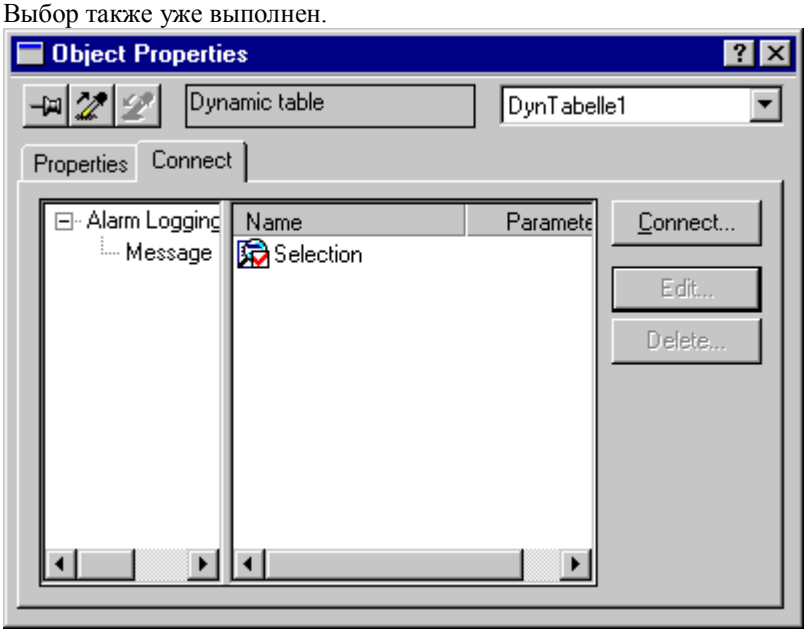
Создать отчет последовательности сообщений. Отчет последовательности сообщений должен автоматически выводиться на печать при заполнении страницы макета.
Этот пример основывается на примере Message Window (Окно Сообщений) (ex_3_chapter_02b.pdl) раздела *Регистрация аварийных сообщений*. В том примере в использующемся окне сообщений уже присутствует кнопка панели инструментов, отвечающая за формирование отчета, а отчет последовательности сообщений активизирован.

Концепция реализации

Существуют готовые системный макет и системное задание печати, удовлетворяющие указанным требованиям. Это макет *@alrtmef.rpl* и задание печати *@Report Alarm Logging RT Message sequence*. Этот макет следует скопировать и внести в него необходимые изменения. В качестве задания печати используется системное задание, при этом изменяется только макет, используемый этим заданием.

Реализация в дизайнере отчетов

Шаг	Процедура: Реализация в дизайнере отчетов
1	В *проводнике WinCC* откройте системный макет *@alrtmef.rpl* нажатием R (правой кнопки мыши) на его имени.  Командой меню *File (Файл)* → *Save As... (Сохранить как...)* системный макет сохраняется с другим именем. В данном примере используется имя *alg_mef.rpl*.
2	Статическая часть отчета содержит верхний и нижний колонтитулы. Элементы статической части могут быть изменены в соответствии с вашими требованиями.

Шаг	Процедура: Реализация в дизайнере отчетов
3	Динамическая часть отчета содержит объект *Dynamic Object (Динамический объект)* → *Dynamic Table (Динамическая таблица)*. В данном примере это объект *DynTable1*. Открывается диалоговое окно свойств объекта *DynTable1* и выбирается закладка *Connect (Соединение)*. Таблица уже присоединена к *Message Sequence Report (Отчету последовательности сообщений)* для *Alarm Logging Runtime (Системы исполнения регистрации аварийных сообщений)*. Выбор также уже выполнен. 
4	По нажатию (мыши) → *Edit (Правка)* или D (двойному щелчку мыши) на пункте *Selection (Выбор)* откроется диалоговое окно для выбора блоков сообщений. В этом окне уже выбраны системные блоки *Date (Дата)*, *Time (Время)*, *Number (Число)* и *Loop in Alarm (Цикл аварийных сообщений)*. В данном примере все остальные блоки сообщений выбраны по нажатию на кнопку, изображенную ниже. Диалоговое окно закрывается по кнопке *OK*. Затем макет сохраняется. 

Изменение задания печати

Шаг	Процедура: Изменение задания печати
1	В *проводнике WinCC* откройте системное задание печати *@Report Alarm Logging RT Message sequence* при помощи D (двойного щелчка) или R (правой кнопки мыши) на этом имени. 
2	На закладке *Print Job (Задание печати)* указывается только что созданный макет *Layout alrtmef.rpl*. Кроме того, на закладке *Printer Setup (Настройка принтера)* указывается подходящий принтер. Никаких других действий не требуется. Закройте диалоговое окно нажатием на *OK*.
3	В редакторе *Alarm Logging (Регистрация аварийных сообщений)* необходимо активизировать отчет последовательности сообщений. Для этого откройте редактор. После нажатия R (правой кнопки мыши) на пункте *Reports (Отчеты)* откроется диалоговое окно *Assigning Report Parameters (Установка параметров отчета)*.  В этом окне выставляется *флажок* отчета последовательности сообщений.
4	Если пользователь не запускает задание печати вручную, отчет последовательности сообщений будет выведен на печать автоматически, как только заполнится страница макета. Для того чтобы предоставить пользователю возможность запустить отчет в любой момент, во время настройки макета окна сообщений на нем нужно поместить специальную кнопку. Это кнопка *Report Functions (Функции отчета)*. В примере проекта эта функция уже выбрана в макете *MessageWindow_04*, использующемся в кадре *ex_3_chapter_02b.pdl*.  Изменения, сделанные в *системе регистрации аварийных сообщений*, нужно сохранить.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

В созданном макете у отчета последовательности сообщений необходимо изменить блоки сообщений.

4.3.7 Вывод отчета последовательности сообщений на строчный принтер

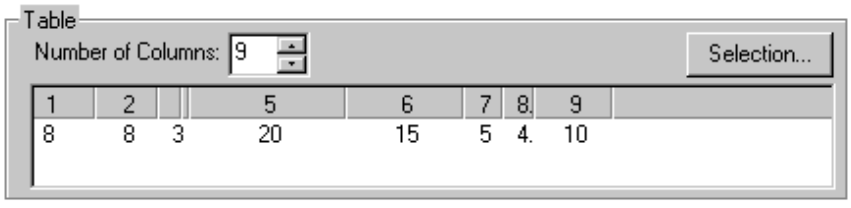
Постановка задачи

Сформировать отчет последовательности сообщений таким образом, чтобы он подходил для вывода его на строчный принтер. При приходе сообщения для отчета оно должно быть напечатано автоматически.

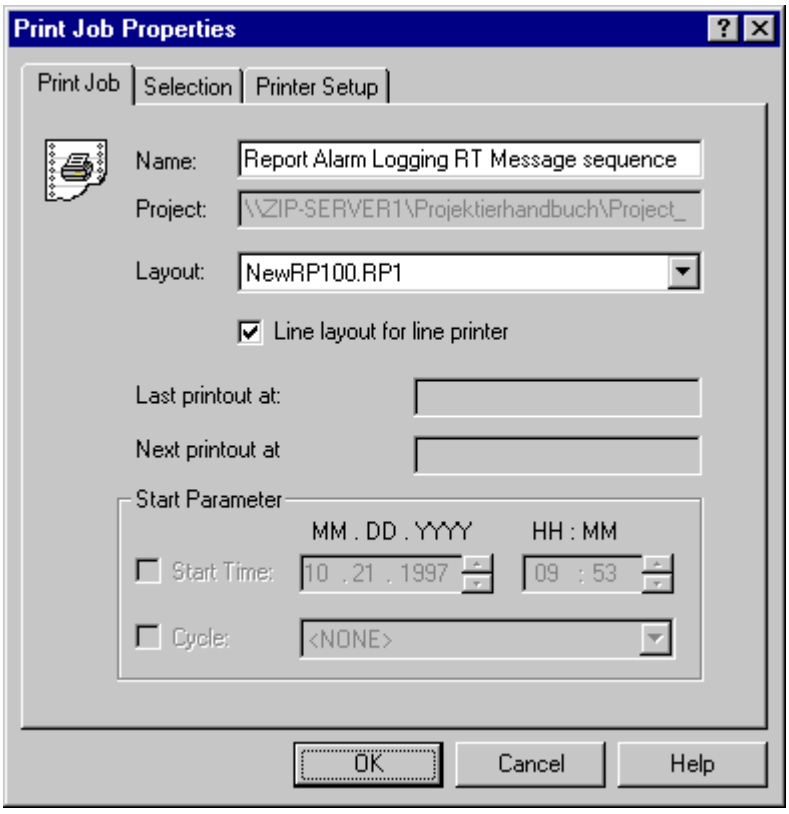
Концепция реализации

Создается строчный макет. Этот макет назначается системному заданию печати *@Report Alarm Logging RT Message sequence*.

Создание строчного макета (Line Layout)

Шаг	Процедура: Создание строчного макета
1	В *проводнике WinCC* щелчком R (правой кнопки мыши) на соответствующем пункте создается новый строчный макет. 
2	Создается новый строчный макет с именем *NewRP100.RP1*. В примере проекта оставлено это имя. По D (двойному щелчку мыши) на имени нового макета в правом окне открывается *Line Layout Editor (Редактор строчного макета)*. В этом редакторе производится общая настройка полей страницы, нижнего и верхнего колонтитулов и т.п. В поле *Table (Таблица)* по щелчку (мыши) → *Selection (Выбор)* открывается диалоговое окно для выбора блоков сообщений в отчете последовательности сообщений. В данном примере выбираются все доступные блоки сообщений.
3	Количество колонок и их ширина устанавливаются автоматически в соответствии с количеством и порядком выбранных блоков сообщений.  Сохраните сделанные изменения и закройте редактор макетов.

Изменение задания печати

Шаг	Процедура: Изменение задания печати
1	В *проводнике WinCC* откройте системное задание печати *@Report Alarm Logging RT Message sequence* при помощи D (двойного щелчка) или R (правой кнопки мыши) на этом имени.
2	На закладке *Print Job (Задание печати)* указывается только что созданный макет *Layout NEWRPL00.rpl* и устанавливается флажок *Line Layout for Line Printer (Строчный макет для строчного принтера)*. Кроме того, на закладке *Printer Setup (Настройка принтера)* указывается подходящий принтер. Никаких других действий не требуется. Закройте диалоговое окно нажатием на *OK*. 
3	В редакторе *Alarm Logging (Регистрация аварийных сообщений)* необходимо активизировать отчет последовательности сообщений. Для этого откройте редактор. После нажатия R (правой кнопки мыши) на пункте *Reports (Отчеты)* откроется диалоговое окно *Assigning Report Parameters (Установка параметров отчета)*. В этом окне выставляется *флажок* отчета последовательности сообщений.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

В редакторе строчных макетов следует изменить параметры страницы и печатаемые блоки сообщений.

4.3.8 Отчет архива сообщений (ex_3_chapter_02c.pdl)

Постановка задачи

Создать отчет архива сообщений. Задание печати должно активизироваться пользователем посредством нажатия на определенную кнопку.
Этот пример основан на примере Message Archiving (Архивация сообщений) (ex_3_chapter_02c.pdl) в разделе *Регистрация аварийных сообщений*. В том примере в окне краткосрочного (short–term) циклического (revolving) архива сообщений уже присутствует кнопка панели инструментов, отвечающая за формирование отчета, а отчет последовательности сообщений активизирован.

Концепция реализации

Существуют готовые системный макет и системное задание печати, удовлетворяющие указанным требованиям. Это макет *@alrtmef.rpl* и задание печати *@Report Alarm Logging RT Revolving archive*. Этот макет следует скопировать и внести в него необходимые изменения. В качестве задания печати используется системное задание, при этом изменяется только макет, используемый этим заданием.

Реализация в дизайнере отчетов

Шаг	Процедура: Реализация в дизайнере отчетов
1	В *проводнике WinCC* откройте системный макет *@alrtmef.rpl* нажатием R (правой кнопки мыши) на его имени. Командой меню *File (Файл)* → *Save As… (Сохранить как…)* системный макет сохраняется под другим именем. В данном примере используется имя *alg_uma.rpl*.
2	Статическая часть отчета содержит верхний и нижний колонтитулы. Элементы статической части могут быть изменены в соответствии с вашими требованиями.
3	Динамическая часть отчета содержит объект *Dynamic Object (Динамический объект)* → *Dynamic Table (Динамическая таблица)*. В данном примере это объект *DynTable1*. Открывается диалоговое окно свойств объекта *DynTable1* и выбирается закладка *Connect (Соединение)*. Таблица уже присоединена к *Short–Term Archive Report (Отчету краткосрочного архива)* для *Alarm Logging Runtime (Системы исполнения регистрации аварийных сообщений)*. Выбор также уже выполнен.
4	По нажатию (мыши) → *Edit (Правка)* или D (двойному щелчку мыши) на пункте *Selection (Выбор)* откроется диалоговое окно для выбора блоков сообщений. В этом окне уже выбраны системные блоки *Date (Дата)*, *Time (Время)*, *Number (Число)*. В данном примере все остальные блоки сообщений выбраны по нажатию на кнопку, изображенную ниже. Диалоговое окно закрывается по кнопке *OK*. Затем макет сохраняется. 

Изменение задания печати

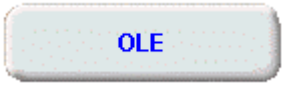
Шаг	Процедура: Изменение задания печати
1	В *проводнике WinCC* откройте системное задание печати *@Report Alarm Logging RT Revolving archive* при помощи D(двойного щелчка) или R (правой кнопки мыши) на этом имени.
2	На закладке *Print Job (Задание печати)* указывается только что созданный макет *Layout alrtuma.rpl*. Кроме того, на закладке *Printer Setup (Настройка принтера)* указывается подходящий принтер. Никаких других действий не требуется. Закройте диалоговое окно нажатием на *OK*.
3	В редакторе *Alarm Logging (Регистрации аварийных сообщений)* необходимо активизировать отчет последовательности сообщений. Для этого откройте редактор. После нажатия R (правой кнопки мыши) на пункте *Reports (Отчеты)* откроется диалоговое окно *Assigning Report Parameters (Установка параметров отчета)*.  В этом окне выставляется *флажок* отчета архива сообщений. Изменения, сделанные в *Системе регистрации аварийных сообщений*, следует сохранить.
4	Для того чтобы предоставить пользователю возможность запустить отчет в любой момент, во время настройки макета окна сообщений на нем нужно поместить специальную кнопку. При применении пользовательской панели инструментов нажатие на эту кнопку следует имитировать при помощи стандартной функции. Это функция *ACX_OnBtnPrint ()*. В данном примере такая кнопка уже присутствует в кадре *ex_3_chapter_02c.pdl*. 

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

В созданном макете следует изменить блоки сообщений для отчета архива сообщений.

4.4 Связь с EXCEL с использованием OLE



В режиме исполнения примеры, имеющие отношение к этой теме, доступны по нажатию (мышью) на *кнопке*, изображенной выше. Примеры приведены в кадре *ex_3_chapter_04.pdl* и книге Excel *OLE_Communication.xls*.

OLE (Object Linking and Embedding) — до 1996 года – общее название группы объектно–ориентированных технологий Microsoft на основе COM (OLE 1, OLE 2, OLE automation, OLE Database и др.); с 1996 года после введения термина ActiveX применяется для обозначения технологий на основе COM, используемых для создания составных документов внедрением и связыванием

4.4.1 Чтение и запись значений тегов (ex_3_chapter_04.pdl)

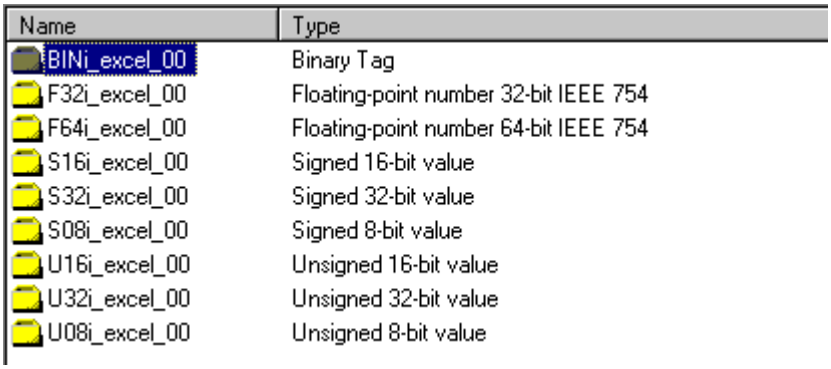
Постановка задачи

Записать значения внутренних тегов различных типов в таблицу Excel. Во вторую колонку электронной таблицы нужно ввести значения уставок для этих тегов. После этого введенные значения следует записать обратно в проект WinCC.

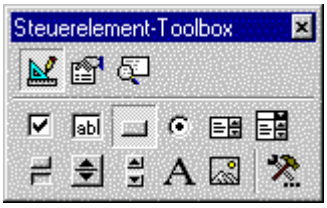
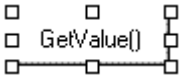
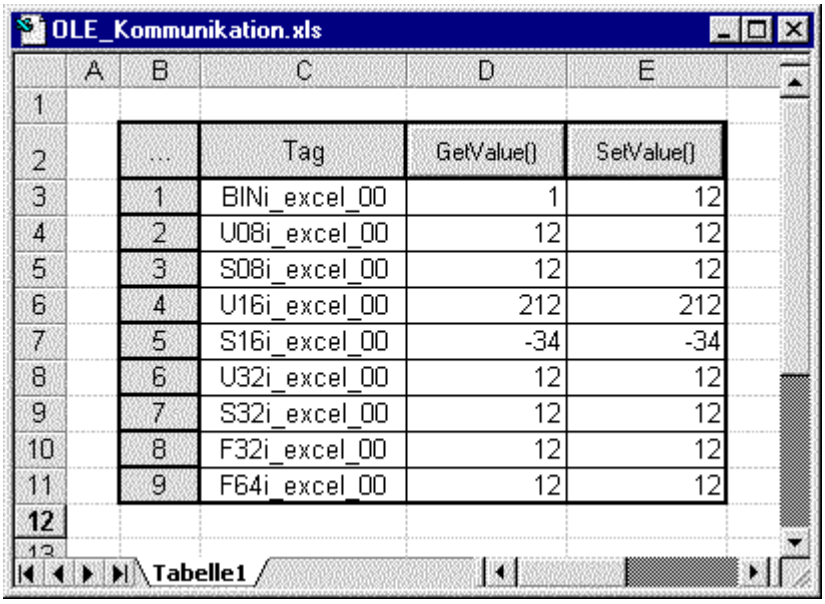
Концепция реализации

В кадре для отображения текущего и ввода нового значения каждого тега создается по одному полю ввода/вывода.
В Excel (версии 8.0) создается электронная таблица (лист). В колонку этой таблицы вводятся названия тегов, которые следует считывать и записывать. К таблице добавляются две кнопки. К ним подключаются 2 макроса, считывающие имена тегов подлежащих обработке, и получающие их значения или записывающие соответствующие уставки.

Реализация в проекте WinCC

Шаг	Процедура: Реализация в проекте WinCC
1	В менеджере тегов создается несколько тегов различных типов. В примере используются следующие теги: 
2	Для каждого тега создается *Smart Object (Интеллектуальный объект)* → *I/O Field (Поле Ввода/Вывода)*. Для атрибута каждого тега *Properties (Свойства)* → *Output/Input (Вывод/ввод)* → *Output Value (Выходное значение)* создается *связь с тегом*.

Реализация в Excel (версия 8.0)

Шаг	Процедура: Реализация в Excel
1	Создайте новый файл Excel. В данном примере файл называется *OLE_Communication.xls*. В электронной таблице заполните столбец именами тегов, которые будут обрабатываться.
2	Создайте кнопку для чтения значений тегов. С этой целью при помощи команды меню *View (Вид)* → *Toolbars (Панели инструментов)* отобразите панель *Control Toolbox (Управляющий инструментарий)*, если она не активна. Выберите элемент *Command Button (Управляющая кнопка)* и поместите его в таблицу. 
3	Свойства этого элемента можно задать в диалоговом окне, которое вызывается нажатием на изображенную ниже кнопку. В примере используется имя объекта *GetValue* и надпись *GetValue ()*. 
4	После D (двойного щелчка мыши) на кнопке откроется редактор Visual Basic, в котором можно задать выполняемую функцию. 
5	Таким же образом создайте другую кнопку для записи значений тегов. 

	A	B	C	D	E
1					
2		...	Tag	GetValue()	SetValue()
3		1	BINi_excel_00	1	12
4		2	U08i_excel_00	12	12
5		3	S08i_excel_00	12	12
6		4	U16i_excel_00	212	212
7		5	S16i_excel_00	-34	-34
8		6	U32i_excel_00	12	12
9		7	S32i_excel_00	12	12
10		8	F32i_excel_00	12	12
11		9	F64i_excel_00	12	12
12					

Tabelle1

Процедура для чтения значений тегов

```
Rem Read Tag Values in WinCC-Project

Private Sub GetValue_Click()

    Dim mcp As Object
    Dim var As String
    Dim value As Variant
    Dim cell As Variant
    Dim i As Integer

    Set mcp = CreateObject("WinCC-Runtime-Project")

    Cell = "C3"
    i = 1

    Do While Not Range(cell) = ""
        var = Range(cell)
        value = mcp.GetValue(var)
        Range("D" & 2 + i).value = value
        cell = "C" & 3 + i
        i = i + 1
    Loop

End Sub
```

Генерируется объект WinCC, представленный переменной *mcp*.

Содержимое ячеек колонки, хранящей имена тегов, считывается в цикле. В проекте WinCC значения тегов считаются при помощи функции *GetValue ()* и записываются в следующую колонку. Цикл продолжается до тех пор, пока не будет найдена пустая ячейка.

Процедура для записи значений тегов

```
Rem Set Tag Values in WinCC-Project

Private Sub SetValue_Click()

    Dim mcp As Object
    Dim var As String
    Dim value As Variant
    Dim cell As Variant
    Dim i As Integer
    Dim bRet As Integer

    Set mcp = CreateObject("WinCC-Runtime-Project")

    Cell = "C3"
    i = 1

    Do While Not Range(cell) = ""
        var = Range(cell)
        value = Range("E" & 2 + i).value
        bRet = mcp.SetValue(var, value)
        cell = "C" & 3 + i
        i = i + 1
    Loop

End Sub
```

Генерируется объект WinCC, представленный переменной *mcp*.

Содержимое ячеек колонки, хранящей имена тегов, считывается в цикле. Дополнительно считываются ячейки колонки, содержащей значения тегов. Теги записываются в проект WinCC с помощью функции *SetValue ()*. Цикл продолжается до тех пор, пока не будет найдена пустая ячейка.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

Обмен данными между WinCC и Excel производится с помощью функций GetValue() и SetValue(). В Excel данные можно обработать произвольным образом.

4.5 Дополнительные элементы примеров

В этом разделе описываются дополнительные элементы, использующиеся в некоторых кадрах. Их описание в соответствующих примерах привело бы к излишней детализации, т.к. они напрямую не относятся к рассматриваемым вопросам. Эта глава завершает описание проекта–примера.

4.5.1 Индекс кадров

Постановка задачи

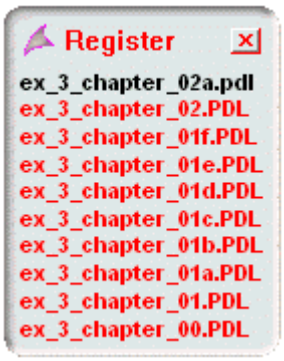
Сохранять порядок отображения 10 последних выбранных кадров. Необходимо предоставить возможность листать кадры в обратном порядке по кнопке back. Кнопка forward должна перемещать пользователя один кадр очереди вперед. В отдельном кадре следует в правильном порядке отобразить все кадры индекса. Необходимо также предоставить возможность прямого выбора любого из кадров индекса.

Концепция реализации

Порядок кадров сохраняется в 10 статических переменных Си функции проекта. Эта функция вызывается каждый при смене кадра. Выбор кадра также осуществляется при помощи кнопок back и forward и прямого указания кадра.

Реализация в графическом дизайнере

Шаг	Процедура: Реализация в графическом дизайнере
1	Используются два тега типа *Binary Tag (Двоичный тег)*. Это теги *BINi_ex_org_00* и *BINi_ex_org_01*. Они используются для реализации функций кнопок back и forward. Кроме того, используется тег типа *Unsigned 16–Bit Value (16–битная величина без знака)*. В данном примере это тег *U16i_ex_org_00*. В нем хранится текущая позиция в очереди кадров. Также используется тег типа *Text Tag 16–Bit Character Set (Текстовый тег 16–битного набора символов)*. В данном примере это тег *T16x_ex_org_00*. В него записывается имя текущего кадра.
2	Для управления последовательностью кадров имеется готовая функция проекта — *CreatePictureSequence*. Она вызывается всякий раз, когда происходит смена кадра. Это делается с помощью *процедуры Си*, созданной для события *Events (События)* → *Property Topics (Разделы свойств)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* объекта *workspace (рабочая область)* в кадре *ex_0_startpicture_00.pdl*. Каждый раз при вызове этой функции имя нового кадра сохраняется в индексе, а имена других кадров сдвигаются на одну позицию назад.
3	В кадре *ex_2_keyboard_00.pdl* создаются два управляющих элемента для движения вперед и назад. При нажатии на любой из этих объектов производится вызов функции *CreatePictureSequence*. После этого данная функция выполняет смену кадров. Используя атрибуты *Smart Objects (Интеллектуальный объект)* → *Graphic Objects (Графические объекты)*, эти два элемента управления можно блокировать. 

Шаг	Процедура: Реализация в графическом дизайнере
4	Приведенная ниже кнопка вызывает кадр, который отображает текущий индекс. В данном примере это кадр *ex_9_register_00.PDL*. 
5	На этом кадре создается 10 объектов типа *Standard Objects (Стандартные объекты)* → *Static Texts (Статический текст)*. У всех объектов удаляется текст, присутствующий по умолчанию. Для атрибута кадра *Properties (Свойства)* → *Geometry (Геометрия)* → *Picture Width (Ширина кадра)* создается *процедура Cu*, вызывающая функцию проекта *CreatePictureSequence*. Эта процедура заносит в *статические тексты* сохраненные имена кадров. Данная *процедура Cu* запускается по изменению тега *T16x_ex_org_00*. Это приводит к обновлению изображения при выборе кадра. У каждого *статического текста* для события *Events (События)* → *Mouse (Мышь)* → *Press left (Нажатие левой кнопки)* создается *процедура Cu*, которая вызывает функцию проекта и выполняет замену текущего кадра выбранным. Название выбранного кадра подсвечивается. Это делается с использованием *Dynamic Dialog (Динамического диалога)* для атрибутов *Properties (Свойства)* → *Colors (Цвета)* → *Font Color (Цвет шрифта)* каждого *статического текста*. 
6	По переданному параметру функция проекта получает информацию о том, из какой позиции она была вызвана. Для этого параметра в файле *APDEFAP.H* определены различные константы. Этот файл находится в папке *Library* каталога проекта. Определены следующие константы: #define REG_INSERTPICTURE 0 #define REG_BACK 1 #define REG_FORWARD 2 #define REG_DIRECT 3 #define REG_SHOWREGISTER 4

Функция проекта для управления последовательностью кадров

```
#include "APDEFAP.H"
#define MAX_REG 10
void CreatePictureSequence(char* PicName,int nFlag,int nPos)
{
static char PictureName[MAX_REG][40] = {"","","","","","","","","",""};
int i;
static int pos = 0;
static int st = 0;
static int biz = 0;

if (nFlag==REG_INSERTPICTURE){
 if (st == 0 ){
  pos = 0;
  if (biz < MAX_REG) biz++;
  for ( i=(MAX_REG-1) ; i>0 ; i-- ){
   strcpy(PictureName[i],PictureName[i-1]);
   }
  strcpy(PictureName[0],PicName);
  }
 else st=0;
 }
if (nFlag==REG_BACK){
 pos++;
 if ( pos > (MAX_REG-1) ) pos=(MAX_REG-1);
 st = 1;
 SetPictureName("ex_0_startpicture_00.PDL",
                "workspace",PictureName[pos]);
 }
if (nFlag==REG_FORWARD){
 pos--;
 if ( pos < 0 ) pos=0;
 st = 1;
 SetPictureName("ex_0_startpicture_00.PDL",
                "workspace",PictureName[pos]);
 }
if (nFlag==REG_SHOWREGISTER){
 SetText("ex_9_register_00.PDL",
         "Static Text1",PictureName[0]);
 SetText("ex_9_register_00.PDL",
         "Static Text2",PictureName[1]);
 SetText("ex_9_register_00.PDL",
         "Static Text3",PictureName[2]);
 SetText("ex_9_register_00.PDL",
         "Static Text4",PictureName[3]);
 SetText("ex_9_register_00.PDL",
         "Static Text5",PictureName[4]);
 SetText("ex_9_register_00.PDL",
         "Static Text6",PictureName[5]);
 SetText("ex_9_register_00.PDL",
         "Static Text7",PictureName[6]);
 SetText("ex_9_register_00.PDL",
         "Static Text8",PictureName[7]);
 SetText("ex_9_register_00.PDL",
         "Static Text9",PictureName[8]);
 SetText("ex_9_register_00.PDL",
         "Static Text10",PictureName[9]);
 }
if (nFlag==REG_DIRECT){
 st=1;
 pos=nPos;
 }
if ((nFlag!=REG_SHOWREGISTER) && (nFlag!=REG_DIRECT)){
 if (pos<(biz-1))      SetTagBit("BINi_ex_org_00",FALSE);
 else                  SetTagBit("BINi_ex_org_00",TRUE);

 if (pos>0)            SetTagBit("BINi_ex_org_01",FALSE);
 else                  SetTagBit("BINi_ex_org_01",TRUE);
 }
SetTagWord("U16i_ex_org_00",(WORD)pos);
}
```

Если параметр *nFlag* имеет значение *REG_INSERTPICTURE*, то функция была вызвана из *процедуры Cи* для события *Events (События)* → *Property Topics (Разделы свойств)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра) workspace (рабочей области)* в кадре *ex_0_startpicture_00.pdl*. Рабочая область *workspace* — это окно кадра, в котором отображаются все кадры примера. Если смену кадров не следует вносить в индекс, то при предшествующем вызове функции тег *st* нужно установить в *1*. Сам индекс состоит из статического массива 10 текстовых тегов.

Если параметр *nFlag* имеет значение *REG_BACK*, то была нажата кнопка back. Функция выполняет смену кадра, но в индекс он не добавляется.

Если параметр *nFlag* имеет значение *REG_FORWARD*, то была нажата кнопка forward. Функция выполняет смену кадра, но в индекс он не добавляется.

Если параметр *nFlag* имеет значение *REG_SHOWREGISTER*, производится обновление всех *статических текстов* в кадре *ex_9_register_00.pdl*. Этот случай соответствует выбору кадра из индекса или смене кадра при открытом окне индекса.

Если параметр *nFlag* имеет значение *REG_DIRECT*, то кадр был выбран непосредственно с использованием *статического текста*. *Процедура Cи статического текста* выполняет смену кадра, но в индекс он не добавляется.

Замечание относительно основных применений

В общем случае для применения описанного метода необходимо внести следующие изменения:

Приведенная конфигурация полностью готова к использованию. Для ее функционирования необходимо создать 5 используемых тегов, добавить элементы управления и функцию проекта.

Если прямой выбор кадра и отображение индекса не требуются, то в функции проекта фрагменты *REG_DIRECT* и *REG_SHOWREGISTER* могут быть опущены.

Для изменения количества запоминаемых кадров нужно изменить значение максимального номера кадра *MAX_REG* в функции проекта.

4.5.2 Индекс

Индекс проекта–примера доступен по нажатию (мышью) на кнопку, изображенную выше.

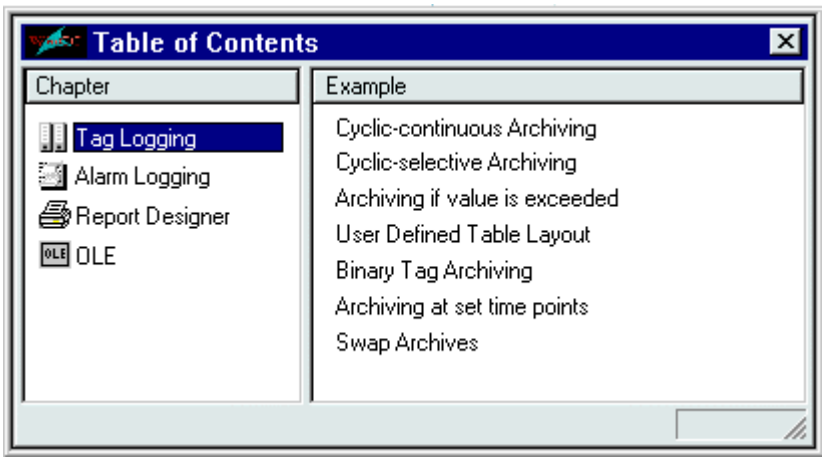
Постановка задачи

Отобразить индекс проекта при помощи диалога. В одном окне следует перечислить основные разделы. В другом окне — отображать примеры, относящиеся к выбранному разделу.
Необходимо предоставить возможность прямого выбора примера. Такой выбор должен производиться двойным щелчком.

Концепция реализации

Выбор диалога индекса осуществляется при помощи кнопки панели обзора. Диалог отображается в окне кадра. Он содержит дополнительное окно кадра, в котором отображается содержимое выбранного раздела.

Реализация в графическом дизайнере

Шаг	Процедура: Реализация в графическом дизайнере
1	Используется два тега типа *Unsigned 16–Bit Value (16–битная величина без знака)*. В данном примере это теги *U16i_ex_con_00* и *U16i_ex_con_01*. Эти теги содержат номер выбранного раздела и номер примера.
2	Для отображения описанного диалога существует готовый кадр, называющийся *ex_9_register_01.pdl*. Для каждого раздела создается *Standard Object (Стандартный объект)* → *Static Text (Статический текст)* и *Smart Object (Интеллектуальный объект)* → *Graphic Object (Графический объект)*. Первоначально в окне выбора не выделен ни один раздел: тег *U16i_ex_con_00* имеет нулевое значение. Если (мышью) производится выбор *статического текста*, то в этот тег записывается соответствующий номер раздела. Используя различные *динамические диалоги*, можно изменить цвет выделенного *статического текста*. 

Шаг	Процедура: Реализация в графическом дизайнере
3	Для каждого раздела создается отдельный кадр, который в зависимости от выбранного раздела отображается в *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. Если ни один раздел не выбран, то окно кадра не отображается. Для каждого примера (соответствующего своему разделу) создается *Standard Object (Стандартный объект)* → *Static Text (Статический текст)*. При выборе раздела изначально не отображается ни один пример. При выборе *статического текста* (мышью) номер соответствующего примера записывается в тег *U16i_ex_con_01*. Используя различные *динамические диалоги*, можно изменить цвет выбранного *статического текста*.
4	Для реализации выбора кадра двойным щелчком необходимо создать три внешних переменных Си (перечислены ниже). Они создаются в функции проекта *CreateExternal*. Эта функция выполняется один раз при запуске проекта. Extern BOOL bPress1, bPress2 Extern int nButtonID В *процедуре Си* для атрибута кадра *Properties (Свойства)* → *Geometry (Геометрия)* → *Picture Width (Ширина кадра)* раз в 500 ms запрашивается, был ли сделан двойной щелчок. Если это так, то в зависимости от значения переменной *nButtonID* выполняется определенное действие.

Процедура Си для текста примера Sample Text

```
#include "apdefap.h"
void OnLButtonDown(char* lpszPictureName, char* lpszObjectName, char* lpszP
{
extern BOOL bPress1,bPress2;
static BOOL bToggle = FALSE;
extern int nButtonID;

nButtonID=1;

if (bToggle) bPress1=TRUE;
else bPress2=TRUE;

bToggle=!bToggle;

SetTagWord("U16i_ex_cont_01",(WORD)nButtonID);

}
```

Во внешнюю переменную Си записывается идентификационный номер *статического текста*. Этот номер используется для определения действий, которые нужно выполнить.

При любом событии для мыши *bPress1* и *bPress2* поочередно устанавливаются в *TRUE*.

Процедура Си для определения двойного щелчка

```
#include "apdefap.h"
 long _main(char* lpszPictureName, char* lpszObjectName, char* lpszProperty
{
extern BOOL bPress1,bPress2;
extern int nButtonID;

if ((bPress1)&&(bPress2))
 switch(nButtonID){
 case1:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01.PDL");
       break;
 case2:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01a.PDL");
       break;
 case3:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01b.PDL");
       break;
 case4:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01c.PDL");
       break;
 case5:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01d.PDL");
       break;
 case6:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01e.PDL");
       break;
 case7:SetPictureName("ex_0_startpicture_00.PDL",
                     "workspace","ex_3_chapter_01f.PDL");
       break;
 case8:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01g.PDL");
       break;
 case9:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01i.PDL");
       break;
 case10:SetPictureName("ex_0_startpicture_00.PDL",
                      "workspace","ex_3_chapter_01j.PDL");
        break;
 default:break;
 }

bPress1=FALSE;
bPress2=FALSE;

return 242;
}
```

Опрашиваются обе внешних Си–переменных. Если значение обеих *TRUE*, за последние 500 мс был произведен двойной щелчок на *статическом тексте*. *Процедура Си* выполняется каждые 500 мс, и после каждого вызова обе переменных устанавливаются в *FALSE*.

При распознавании двойного щелчка выполняется действие, зависящее от значения *nButtonID*. Эта переменная ссылается на выбранный *статический текст*.

Возвращается ширина кадра.

4.5.3 Диалоговые окна выбора цвета (ex_3_chapter_01c)

Диалоговые окна выбора цвета, описанные в данном примере, вызываются при нажатии (мышью) на изображенную выше кнопку в кадре *ex_3_chapter_01c*.

Постановка задачи

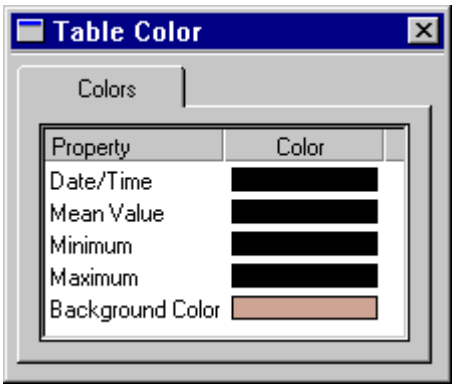
Используя различные диалоговые окна изменить цвета, заданные для таблицы, описанной в примере User–Defined Table Layout (Определяемый пользователем формат таблицы) (ex_3_chapter_01c.pdl) раздела *Регистрация тегов*. Необходимо иметь возможность редактирования как цвета шрифта, так и цвета фона каждого столбца.

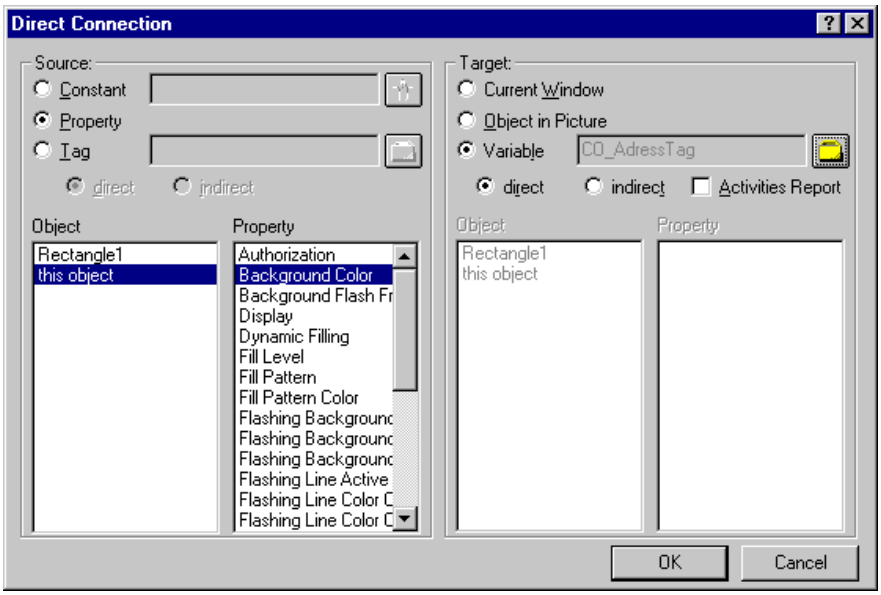
Концепция реализации

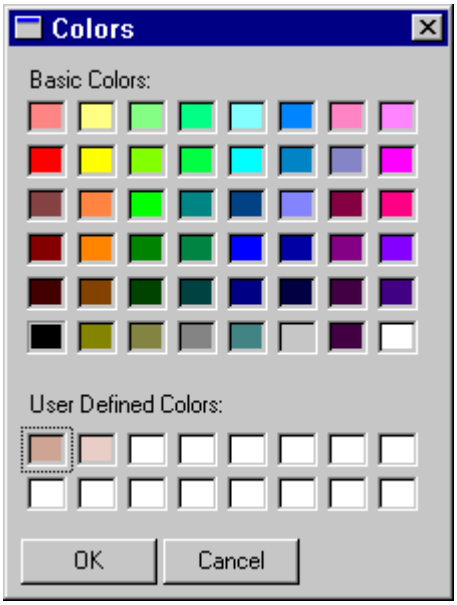
Диалоговые окна выбора цвета определяются как совокупность трех кадров. Первый кадр, вызываемый при нажатии на изображенную выше кнопку, содержит диалоговое окно, показывающее текущие цвета. В этом окне для любого модифицируемого свойства D (двойным щелчком мыши) можно вызвать диалог выбора цвета, содержащий основные 16 цветов. В свою очередь из этого диалогового окна по нажатию кнопки можно вызвать другой диалог, содержащий 50 цветов.

Реализация в графическом дизайнере

Шаг	Процедура: Реализация в графическом дизайнере
1	Используются пять тегов типа *Unsigned 32–Bit Value (32–битная величина без знака)*. В данном примере это теги *CO_TIME*, *CO_MAX*, *CO_MIN*, *CO_MEAN* и *CO_BACK*. В них хранятся текущие значения цветов. Дополнительно используется еще один тег типа *Unsigned 32–Bit Value (32–битная величина без знака)*. В данном примере это тег *CO_TEMP*. Этот тег служит для запоминания цвета, который может быть выбран нажатием на *кнопку OK*. Тег типа *Text Tag 16–Bit Character Set (Текстовый тег 16–битного набора символов)* используется в качестве адресного тега. Он хранит имя тега цвета, с которым ведется работа. В данном примере это *CO_AdressTag*.
2	Таблица, приведенная в кадре *ex_3_chapter_01c.pdl*, состоит из множества простых объектов. Все модифицируемые свойства этих объектов имеют *соединения с тегами*, представляющими цвета.

Шаг	Процедура: Реализация в графическом дизайнере
3	Макет первого диалогового окна был реализован в отдельном кадре. Этот кадр называется *ex_10_FD_00.pdl*. Для каждого модифицируемого свойства создается *Standard Object (Стандартный объект)* → *Static Text (Статический текст)* и *Standard Object* → *Rectangle (Прямоугольник)*. Прямоугольник показывает текущие установки цвета для каждого свойства. Это делается посредством *связи тегов* с соответствующими цветовыми тегами. В *процедуре Cи*, вызываемой для события *Events (События)* → *Mouse (Мышь)* → *Press left (Нажатие левой кнопки)* объектов *Rectangles (Прямоугольники)* и *Static Texts (Статические тексты)*, имя соответствующего тега цвета записывается в адресный тег. По двойному щелчку на прямоугольниках открывается диалог из 16 цветов. Двойной щелчок опрашивается у *Graphic Object1* в атрибуте *Properties (Свойства)* → *Geometry (Геометрия)* → *Position X (Координата X)*. 

Шаг	Процедура: Реализация в графическом дизайнере
4	Макет второго диалогового окна реализован в другом кадре. Это кадр *ex_10_FD_01.pdl*. Для каждого выбираемого цвета создается *Standard Object (Стандартный объект)* → *Rectangle (Прямоугольник)*, фон которого соответствует устанавливаемому цвету. У каждого прямоугольника для события *Events (События)* → *Mouse (Мышь)* → *Press left (Нажатие левой кнопки)* создается *прямое соединение*. Это соединение присваивает атрибуту соответствующего объекта *Background Color (Цвет фона)* цвет тега, указываемого адресным тегом. 
5	У каждого присутствующего в текущем окне *прямоугольника* для события *Events (События)* → *Mouse (Мышь)* → *Mouse Action (Действие мыши)* создается *прямое соединение*. С помощью специального объекта *Windows Object (Объект Windows)* → *Button (Кнопка)* можно открыть следующее диалоговое окно. 
6	Макет третьего диалогового окна реализован в отдельном кадре. Это кадр *ex_10_FD_03.pdl*. Как описано в шаге 4, для каждого выбираемого цвета создается *Standard Object (Стандартный объект)* → *Rectangle (Прямоугольник)*. *Прямое соединение* для события *Events (События)* → *Mouse (Мышь)* → *Press left (Нажатие левой кнопки)*, однако, описывает тег цвета, указанный не в адресном теге, а в теге текущего цвета *CO_TEMP*. Значение, содержащееся в этом теге, записывается в тег цвета только после

Шаг	Процедура: Реализация в графическом дизайнере
	нажатия на кнопку OK. В кадр поместите *Smart Object (Интеллектуальный объект)* → *Graphic Object (Графический объект)*, в котором будет отображаться представленный тегом *CO_TEMP* цвет. В данном примере это объект *Selection*. Координаты этого объекта изменяются при выборе прямоугольника в *процедуре Cu*, связанной с событием *Events (События)* → *Mouse (Мышь)* → *Press left (Нажатие левой кнопки)*. 
7	В кадре *ex_3_chapter_01c.pdl* для каждого диалога сконфигурирован объект *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*. *Окно кадра*, расположенное в первом диалоге, открывается при помощи объекта *Windows Object (Объект Windows)* → *Button (Кнопка)*. 

4.5.4 Окно гистограммы (ex_3_chapter_01e)

Окно гистограммы, описываемое здесь, используется в примере Archiving at Defined Times (Архивация в заданные моменты времени) (ex_3_chapter_01e.pdl) раздела *Регистрация тегов*.

Постановка задачи

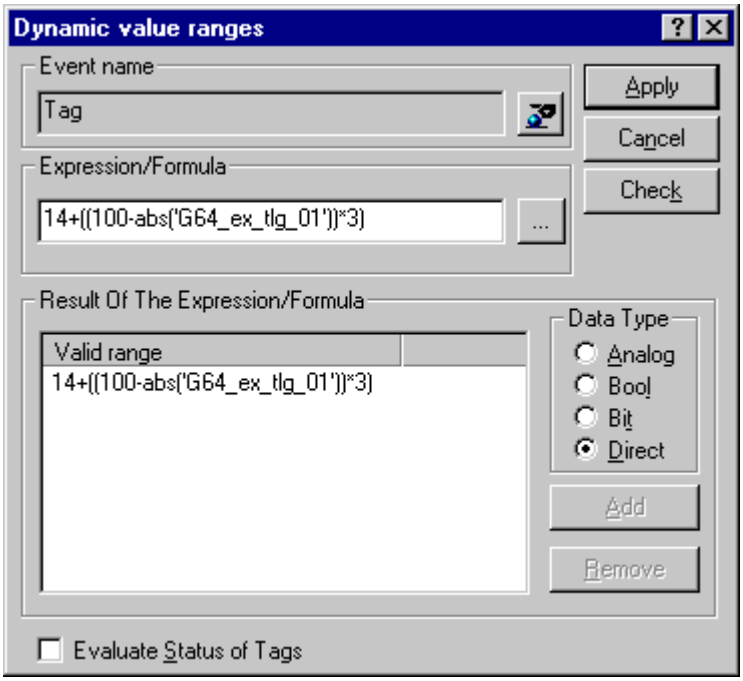
Используя три гистограммы, показать текущие значения тегов, архивируемых в соответствующем кадре. Каждая гистограмма должна активизироваться индивидуально по нажатию кнопки.

Концепция реализации

Каждая гистограмма состоит из объекта *Smart Object (Интеллектуальный объект)* → *Status Display (Индикатор состояния)*, представляющего изображение гистограммы, и объекта *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, в котором отображается фон гистограммы. Ширина *Picture Window (Окна кадра)* при помощи *динамического диалога* изменяется в соответствии со значением отображаемого тега.

Реализация в графическом дизайнере

Шаг	Процедура: Реализация в графическом дизайнере
1	Имеется три кадра, содержащие только *Smart Object (Интеллектуальный объект)* → *Graphic Object (Графический объект)*. Это кадры *ex_10_BH_00.pdl*, *ex_10_BH_01.pdl* и *ex_10_BH_02.pdl*. В каждом из *графических объектов* отображаются растровые картинки, представляющие фон гистограммы.
2	В кадре *ex_3_chapter_01e.pdl* создайте три *Smart Objects (Интеллектуальные объекты)* → *Status Displays (Индикаторы состояния)*, которые будут отображать столбцы гистограмм. В случае если гистограмма неактивна, отображаться будет только фоновый рисунок.

Шаг	Процедура: Реализация в графическом дизайнере
3	Над каждым *индикатором состояния* создается *Smart Object (Интеллектуальный объект)* → *Picture Window (Окно кадра)*, в атрибуте *Properties (Свойства)* → *Miscellaneous (Разное)* → *Picture Name (Имя кадра)* которого указаны кадры, описанные в шаге 1. Для атрибута *Properties (Свойства)* → *Geometry (Геометрия)* → *Window Height (Высота окна)* каждого из таких объектов создается *динамический диалог*, который управляет высотой окна кадра в соответствии со значением тега. Отображаются только положительные значения. Следовательно, необходимо вычислить абсолютную величину тега с помощью функции *abs()*. Максимально возможное отображаемое значение — 100. Так как *окно кадра* является фоном гистограммы, для вычисления его высоты следует из максимального значения тега вычесть текущее. Одна единица гистограммы соответствует 3 точкам изображения, следовательно, вычисленную высоту гистограммы следует умножить на три. Сверху у гистограммы оставлено 14 незаполненных точек, их нужно прибавить к высоте кадра. 
4	При помощи трех объектов *Smart Objects (Интеллектуальные объекты)* → *Status Displays (Индикаторы состояния)* панели инструментов гистограмму можно сделать неактивной. Это делается посредством *процедуры Cu*, вызываемой для события *Events (События)* → *Mouse (Мышь)* → *Press left (Нажатие левой кнопки)*. Эта *процедура Cu* инвертирует признак видимости *окна кадра*, переключает кадры гистограмм и изображение самого объекта. 

Предметный указатель