



Керівництво програміста

Зміст

1. Програмування в системі.....	4
1.1. Конструктор.....	4
1.1.1. Дизайнер.....	5
1.1.2. Дизайнер вікон. Компоненти.....	48
1.1.3. Користувальницькі реєстри.....	73
1.2. Конструктор (у web-інтерфейсі).....	80
1.2.1. Налаштування контейнера графічних елементів.....	88
1.2.2. Налаштування графічних елементів інтерфейсу.....	88
1.2.3. Налаштування елемента Спліт панель.....	89
1.2.4. Налаштування елемента Випадаючий список.....	90
1.2.5. Налаштування елемента Поле введення.....	91
1.2.6. Налаштування елементів: Час, Дата, Період по роках, Дата та час, Прапорець, Текстове поле.....	92
1.2.7. Налаштування елемента Радіальні кнопки.....	93
1.2.8. Налаштування елемента Ієрархічний реєстр.....	94
1.2.9. Налаштування елемента Каскадний реєстр.....	94
1.2.10. Налаштування елемента Графік.....	95
1.2.11. Налаштування елемента Вкладення.....	96
1.2.12. Налаштування елемента Картка.....	96
1.3. API	97
1.3.1. Налаштування API.....	98
1.4. Інструментарій програміста	99
1.4.1. Вбудований Редактор програм	99
1.4.2. Скрипти користувача ISpro WEB.....	104
1.4.3. Дизайнер FastReport.....	116
1.4.4. Засоби створення вихідних форм	122
1.4.5. Дизайнер FastReport .NET Core.....	131
1.4.6. Використання каталогу USER_DATA	132

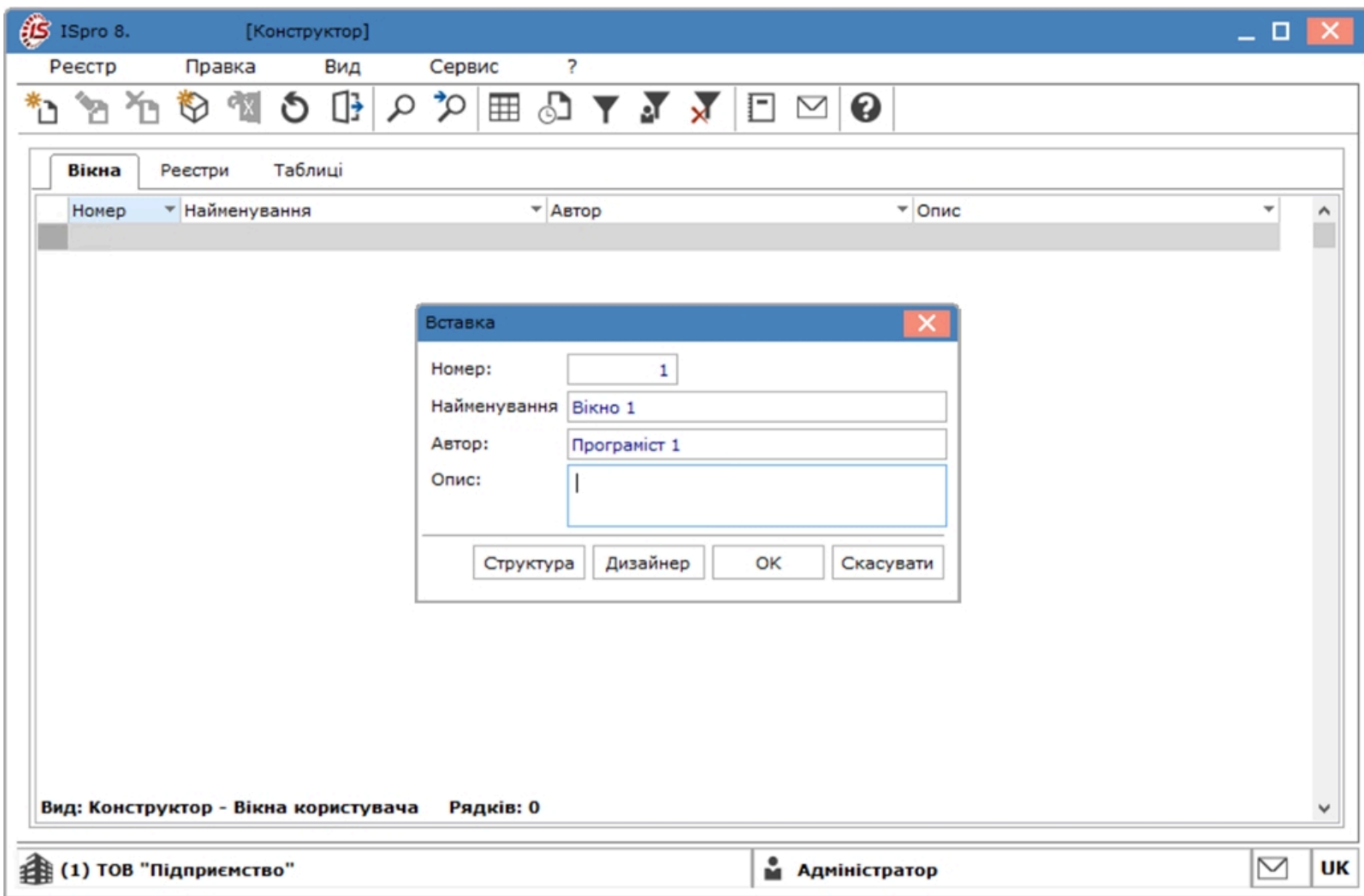
1.4.7. Експорт звітності.....	133
1.4.8. Імпорт файлів MS Office Word (*.doc) в FastReport	142
1.4.9. Розробка звітів MS Office (Excel, Word)	143
1.4.10. Робота з COM об'єктами Комплексу.....	158
1.4.11. Використання в програмі користувача розрахункових функцій головної книги....	165
2. Показчик.....	a

1. Програмування в системі

При роботі в **Комплексі** виникає необхідність програмування. Для використання програм в **Комплексі** використовується **Редактор програм**. Редактор програм присутній в програмному способі формування проводок, в калькуляціях документів, в умовах застосування та інших місцях **Комплексу**.

1.1. Конструктор

Модуль **Конструктор** призначений для розробки користувачами **Комплексу** реєстрів, модулів, таблиць зі своєю, особливою, логікою роботи. Модуль знаходиться в системі **Адміністратор**. Також даний модуль продубльовано в системі **Сервіс**.



Модуль **Конструктор**

Вікно модуля **Конструктор** містить вкладки: **Вікна, Реєстри, Таблиці**.

- **Вікна** — створення дизайну свого вікна, написання до нього логіки відображення і обробки даних. На вкладці **Вікна** по пункту меню **Реєстр / Створити** або по клавіші **Ins** виконується створення вікна, де зазначається **Номер, Найменування** вікна, **Автор** та **Опис**. По кнопці **Дизайнер** відкривається дизайнер зі створення інтерфейсу.;
- **Реєстри** - створення призначених для користувача реєстрів, списків, написання логіки для обробки таких дій як **Створити, Редагувати, Видалити, Копіювати, Групова операція, Подія користувача**; На вкладці **Реєстри** по пункту меню **Реєстр / Створити** або по клавіші **Ins** виконується створення реєстру, де зазначається **Номер, Найменування** реєстру, розташування xml-файлу, **Маркер, Форма, Таблиця, ID таблиці, Автор, Опис**, створюються **Програми** (шляхом натиснення на кнопки **Створення, Видалення, Зміни, Копіювання, Групові операції, Події користувача**).
- **Таблиці** - створення власних таблиць, полів, індексів. На вкладці **Таблиці** по пункту меню **Реєстр / Створити** або по клавіші **Ins** виконується створення таблиці, де зазначається **Ідентифікатор, Найменування**, встановлюється перемикач **Монопольна** або **Спільна**, заповнюються вкладки **Поля** та **Індекси**.

1.1.1. Дизайнер

Дизайнер викликається з модуля **Конструктор**, що призначений для розробки користувачами **Комплексу** реєстрів, модулів, таблиць зі своєю, особливою, логікою роботи. Модуль **Конструктор** знаходиться в системі **Адміністратор**.

Дизайнер представляє собою середовище розробки і містить головне меню, панель інструментів, а також:

- палітру компонентів;
- вікно властивостей об'єктів (Інспектор об'єктів);
- вікно форми;
- редактор коду програми;
- структура модуля.

Цей інструмент для створення користувацьких вікон. Дизайнер за своїм виглядом дуже нагадує середовище розробки звітів **FastReport**.

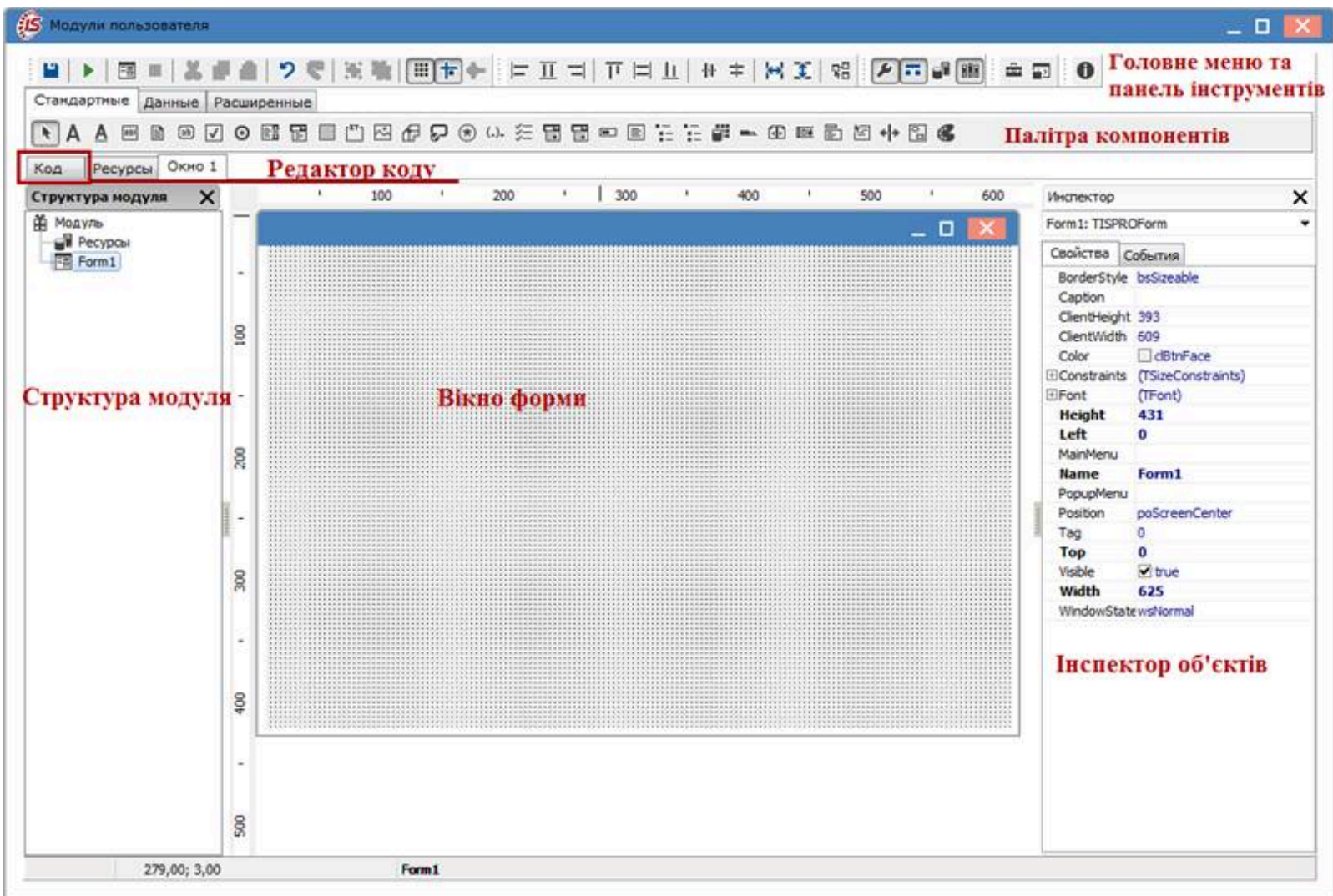


Рис. Модуль Конструктор. Середовище Дизайнера

Головне меню та панель інструментів містять стандартні команди для роботи з проектом.

Таблиця Кнопки панелі інструментів головного меню

Кнопка	Позначення	Комбінація клавіш
	Зберегти	Ctrl+S
	Запуск	Ctrl+P
	Нове вікно	—
	Видалити вікно	—
	Вирізати	Ctrl+X
	Копіювати	Ctrl+C
	Вставити	Ctrl+V

	Видалити вікно	—
	Вирізати	Ctrl+X
	Копіювати	Ctrl+C
	Вставити	Ctrl+V
	Попереднє виправлення	—
	Наступне виправлення	—
	Згрупувати	—
	Розгрупувати	—
	Показати сітку	—
	Вирівнювання по сітці	—
	Розмістити у вузлах сітки	—
	Вирівняти по лівому краю	—
	Відцентрувати по вертикалі	—
	Вирівняти по правому краю	—
	Вирівняти по верхньому краю	—
	Відцентрувати по горизонталі	—
	Вирівняти по нижньому краю	—
	Розташувати по ширині	—
	Розташувати по висоті	—
	Така ж само ширина	—
	Така ж само висота	—
	Встановити порядок обходу	—
	Інспектор об'єктів	F11

	Така ж само висота	—
	Встановити порядок обходу	—
	Інспектор об'єктів	F11
	Структура модуля	—
	Дані	—
	Бібліотека	—
	Вибір доступних ресурсів	—
	Вибір елемента програми	Ctrl+F3
	Про програму	—

Палітра компонентів розташована у головному вікні і має вигляд багатосторінкового блокнота. Кожній сторінці відповідає свій набір компонентів. Дизайнер містить сторінки **Стандартні**, **Дані**, **Розширені**.

Щоб помістити компонент у вікні форми, натисніть один раз на його піктограмі та один раз у потрібному місці форми. Вибраний компонент можна переміщати на формі, а також змінювати розміри, перетягуючи його маркери.

За допомогою **Інспектора** об'єктів можна задавати початкові значення властивостей об'єктів та їхню реакцію на стандартні події. Вікно **Інспектора** об'єктів містить список компонентів поточної форми, а також дві вкладки – **Властивості** та **Події**. Щоб активізувати вікно **Інспектора** об'єктів, використовують клавішу **F11**.

Вкладка **Властивості** складається з двох стовпців: лівий містить назви властивостей компонентів, а правий – їхні значення. Властивості можуть бути простими або комплексними. Комплексні властивості складаються з набору інших властивостей і такі властивості в **Інспекторі** об'єктів позначені символом « + », наприклад  **Font**.

Значення властивостей можна змінювати. Приміром, якщо натиснути на формі (або на іншому компоненті), а тоді обрати її властивість **Caption**, то у вікні зі значенням цієї властивості **Form1** можна ввести новий текст, який і стане заголовком форми.

Властивість **Color** має список можливих значень, який можна відкрити, натиснувши на кнопці  у вікні цієї властивості. Якщо обрати колір **clRed**, то форма чи виокремлений об'єкт набуде червоного кольору. Усю палітру кольорів можна побачити, двічі натиснувши мишею у полі значення кольору.

Всі властивості мають за замовчуванням певні значення при створенні компонента. Приміром, для компонента **Label** властивість **AutoSize** (Автомасштаб) має значення **true** (істина) без додаткових вказівок. Це означає, що розмір компонента підлаштовується під розмір його вмісту. Але можна задавати потрібний розмір за допомогою властивостей **Width** (довжина) та **Height** (висота), якщо при цьому змінити значення властивості **AutoSize** на **false** (хибність).

Деякі з властивостей мають складну структуру. Натиснення миші у полі значень властивості призводить до появи у цьому полі кнопки . Після натиснення цієї кнопки відкриється діалогове вікно, котре і дозволяє встановлювати нові значення цієї властивості. Наприклад, властивість **Font** дозволяє задавати атрибути тексту: шрифт, розмір, стиль та колір літер.

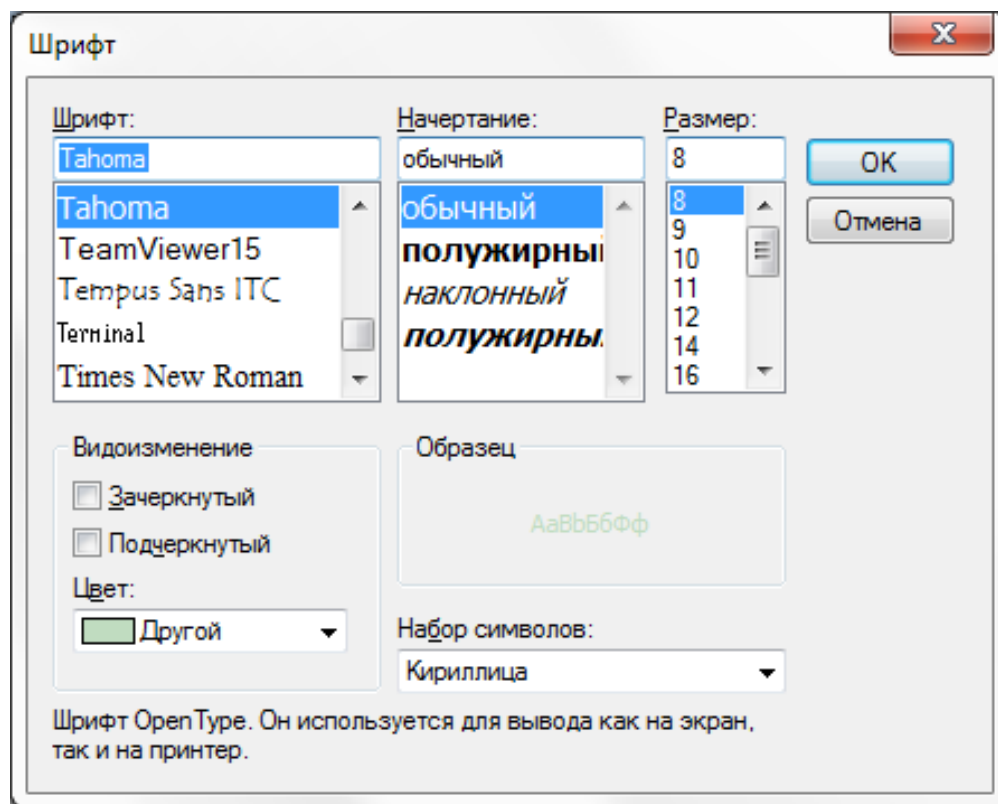


Рис. Властивість **Font**

Якщо на формі треба розмістити кілька однотипних компонентів, то за замовчуванням їхніми іменами буде назва відповідного компонента, наприкінці якого зазначається його порядковий номер на формі. Наприклад, **Button1**, **Button2**, ... чи **Memo1**, **Memo2**, ...

Вкладка **Події** також має два стовпці. У лівому відображаються імена стандартних подій, на які об'єкт може реагувати, а в правому – імена методів-обробників (функцій), які реалізовуватимуть реакцію на подію. Кожній стандартній події відповідає назва методу, яка з'являється після подвійного натиснення мишею у правому стовпці. У цей момент у вікно тексту програми додається шаблон базового коду (функції) для відповідного методу, який треба заповнити.

Багато властивостей і методів є притаманні більшості компонентів, приміром: ім'я, розміри, доступність, видимість тощо. У табл. наведено найчастіше уживані властивості, а в табл. методи функціонування компонентів.

Таблиця 2 - Основні властивості стандартних компонентів

Властивість	Пояснення	Компоненти з такою властивістю
Align	Вирівнювання поля відносно об'єкта, що його містить (форми)	Більшість видимих
AutoSize	Автомасштаб: якщо встановити в 1 (true), розмір компонента автоматично адаптується до розміру символів тексту	Label, Edit, Panel, Image та інші
Caption	Надпис	Button, Label та інші
Color	Колір компонента	Більшість видимих
Count	Кількість рядків списку	Memo, ListBox
Enabled	Доступність до компонента. Визначає, чи реагує компонент на події, пов'язані з мишею, клавіатурою і таймером. Значення false блокує доступ, а сам компонент набуває переважно більш світлого кольору	Усі видимі
Font	Параметри шрифту	Button, Label, Edit, Memo та інші
Glyph	Зображення у форматі bmp, яке можна винести на компонент	BitBtn, SpeedButton
Height	Висота компонента в пікселях	Усі видимі

Glyph	Зображення у форматі bmp, яке можна винести на компонент	BitBtn, SpeedButton
Height	Висота компонента в пікселях	Усі видимі
ItemIndex	Номер обраного елемента списку. Нумерація розпочинається з 0	ListBox
Items	Список можливих значень списку	ListBox, ComboBox, MainMenu та інші
Left	Розміщення лівого краю компонента відносно форми в пікселях. Верхній лівий кут форми має координати 0,0	Усі видимі
Lines	Рядки вікна для введення чи редагування тексту. Нумерація рядків розпочинається з 0	Memo, RichEdit
Name	Ім'я компонента	Усі
ScrollBars	Встановлення у вікні Мемо лінійки прокручування. Можливі значення цієї властивості: None, Vertical, Gorizontal, Both (відсутня, вертикальна, горизонтальна, обидві)	Memo, StringGrid та інші
Sorted	Якщо встановити в 1 (true), усі елементи списку буде відсортовано за зростанням	ListBox, ComboBox
Text	Текст у вікні редагування	Edit, ComboBox, MaskEdit та інші
Top	Розміщення верхнього краю компонента відносно форми в пікселях. Верхній лівий кут форми має координати 0,0	Усі видимі
Visible	Видимість: якщо встановити в 0 (false), компонент стане невидимим на формі	Усі видимі
Width	Ширина компонента в пікселях	Усі видимі

Таблиця – Основні методи стандартних компонентів

Метод	Пояснення	Для яких компонентів застосовується
int Add (AnsiString S)	Долучення рядка S до кінця списку. Метод повертає індекс цього рядка, наприклад: int n=Memo1->Lines->Add(S)	Memo, ListBox, RichEdit та інші
Clear()	Очищення вікна	Edit, Memo та інші
Delete(int n)	Вилучення рядка з номером n, наприклад: Memo1->Lines->Delete(n);	Memo, ListBox, RichEdit та інші
Hide()	Приховання компонента	Більшість видимих
Insert (int n, AnsiString S)	Вставлення рядка S у позицію n, наприклад: Memo1->Lines->Insert(n, S);	Memo, ListBox, RichEdit та інші
LoadFromFile (AnsiString fname)	Вивантаження вмісту із зазначеного файла до поля компонента	Memo, RichEdit та інші
SaveToFile (AnsiString fname)	Зберігання вмісту компонента в файлі із зазначеним ім'ям fname	Memo, RichEdit та інші
SelectAll()	Виокремлення всього тексту	Memo, RichEdit та інші
SetFocus()	Встановлення фокуса (курсору) на цей компонент	Більшість видимих
SetText(char * S)	Долучення тексту S у поточну позицію курсору, наприклад: Memo1->Lines->SetText(S);	Memo та інші
Show()	Показ прихованого компонента	Більшість видимих

Найбільш поширеними у застосуванні подіями є такі:

Таблиця **Події компонентів, що викликаються в Інспекторі**

Подія	Ситуації настання події
OnClick	Натиснення лівою кнопкою миші на об'єкті
OnClickCheck	Зміна користувачем стану індикатора у компоненті CheckListBox
OnChange	Внесення змін до вмісту об'єкта
OnCreate	Подія відбувається при створенні форми
OnEnter	Подія відбувається в момент набуття компонентом фокуса
OnActivate	Подія відбувається при активації форми
OnExit	Подія відбувається в момент втрати компонентом фокуса
OnKeyDown	При натисканні користувачем будь-якої клавіші клавіатури
OnKeyPress	Натиснення символної клавіші клавіатури
OnKeyUp	При відпусканні користувачем будь-якої клавіші клавіатури
OnMouseMove	При переміщенні курсора миші над компонентом
OnStartDrag	Подія відбувається, коли користувач розпочинає перетягування компонента
OnDragDrop	Подія відбувається в момент відпускання перетягнутого компонента над даним компонентом
OnEndDrag	Подія відбувається в момент завершення перетягування компонента
OnMouseWheel	Прокрутка коліщатка миші над елементом
OnMouseWheelDown	Прокрутка коліщатка миші до низу над елементом
OnMouseWheelUp	Прокрутка коліщатка миші вгору над елементом
OnDbClick	Подвійне натиснення лівою кнопкою миші на об'єкті
OnMouseDown	Натиснення кнопки миші над компонентом
OnMouseUp	Відпускання кнопки миші над компонентом

Робочий простір розділено на три області: **Код, Ресурси, Вікно**.

На вкладці **Код** можна писати свою логіку відображення, обробки, збереження даних. **Дизайнер** підтримує мови **Jscript, BasicScript, C++ Script, PascalScript**. Тут так само підтримується всі функції, поля, реєстри і довідники **Комплексу**, які доступні у звітах.

Редактор коду має свою панель інструментів. Застосування методу до певного об'єкта веде до появи заготовки базового коду відповідної функції у вікні **Код**. Заготовка (шаблон) складається із заголовка функції та операторних дужок відповідно до обраної мови програмування. Заготовку заповнює користувач.

Таблиця **Кнопки панелі інструментів** вкладки **Код**

Кнопка	Позначення
	Вибір мови програмування
	Відкрити скрипт
	Зберегти скрипт
	Виконати
	Виконати до поточної позиції
	Відладка
	Зупинити
	Обчислити
	Точка зупинки
	Коментувати / Розкоментувати

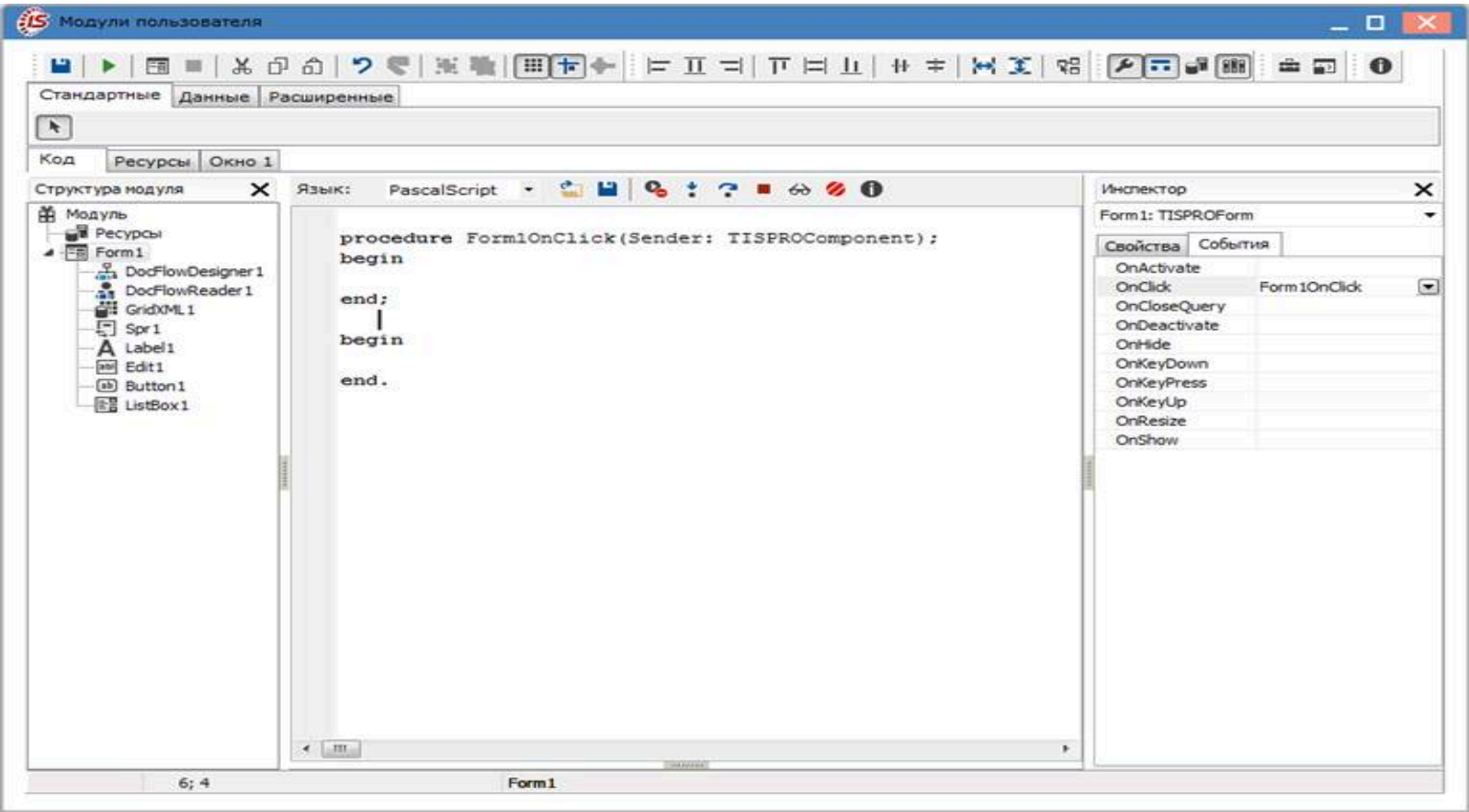


Рис. Дизайнер. Вкладка Код

На вкладці **Ресурси** проводиться робота з запитами, читання даних з **Комплексу**, робота з тимчасовими таблицями, файлами DBF. На робочу область програми можна виносити такі компоненти з вкладок **Стандартні**, **Дані** та **Розширені**.

Таблиця Компоненти вкладки **Стандартні** вкладки **Ресурси**

Компонент	Назва компоненту	Пояснення
	ImageList	Компонент ImageList являє собою набір зображень однакових розмірів, на які можна посилатися за індексами, що починаються з 0.
	ToolbarImageListClassic	Панель з наборами зображень
	ToolbarImageListMetro	Панель з наборами зображень для панелі інструментів
	OpenDialog	Компонент для відкриття файлів
	SaveDialog	Компонент для збереження файлів
	XML	Запускає редактор для XML-документів

	OpenDialog	Компонент для відкриття файлів
	SaveDialog	Компонент для збереження файлів
	XML	Запускає редактор для XML-документів
	Mail	Компонент для роботи з електронними листами

На закладці **Дані** виведені спеціальні компоненти, у властивостях яких, можна вказати Dataset і поле, з якого отримувати дані.

Таблиця Компоненти вкладки **Дані** вкладки **Ресурси**

Компонент	Назва компоненту	Пояснення
	ODBCQuery	Запит. Дозволяє працювати з результатом запиту як з таблицею.
	DBFTable	Таблиця. Дозволяє підключити таблиці бази даних.
	TmpDbTable	Тимчасова таблиця
	ADODatabase	Використовується для вказівки бази даних і роботи з транзакціями.
	ADOTable	Таблиця доступна через ADO
	ADOQuery	запит до бази даних. Це може бути як запит, в результаті якого повертаються дані і бази (наприклад, SELECT), так і запит, що не повертає даних (наприклад, INSERT).

4 Компоненти, що починаються з букв **ADO** використовуються для зв'язку з базами даних через Active Data Objects (ADO) - це технологія стандартного звернення до реляційних даних від Microsoft.

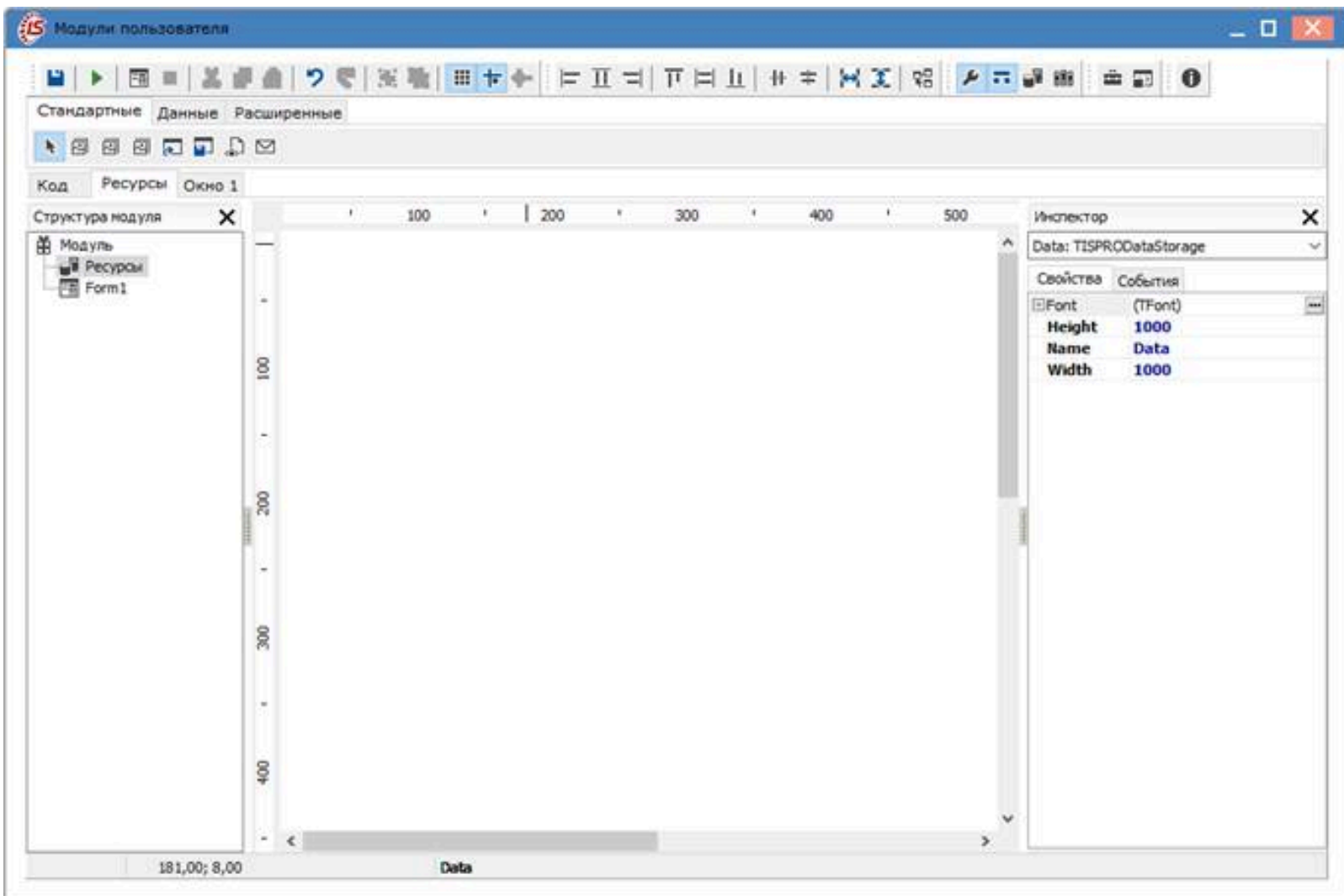


Рис. **Дизайнер**. Вкладка **Ресурси**

На вкладці **Вікно** можна створювати свій дизайн вікна, виводити на форму такі стандартні елементи такі, як **Label**, **Edit**, **Button**, **Memo**, **RadioButton**, **ListView** та інші.

Форма – це вікно Windows, яке утворюється в одному з можливих для вікон стилів. Увесь внутрішній простір є робочою областю, яка має сітку вирівнювання для зручного розташування компонентів на формі. Для виконання групових операцій декілька компонентів можна об'єднувати (кнопки на панелі інструментів головного меню **Групувати / Розгрупувати**). Можна також додати або видалити (клавішею **Del**) окремий компонент. Переміщення виокремленого компонента в межах форми виконується за допомогою правої кнопки миші. Над компонентами та їхніми групами можна виконувати операції вирізання, копіювання в буфер обміну та вставляння з буферу.

Вирівнювати компоненти можна як відносно вікна форми, так і один відносно одного. Для цього використовуються відповідні команди-кнопки з панелі інструментів.

Таблиця Компоненти вкладки **Стандартні**

Компонент	Назва компоненту	Пояснення
	Курсор	Дозволяє рухати та змінювати розмір компонентів, що розміщені на формі
	Label	Напис. Створює на формі текстову мітку або напис.
	LinkLabel	Гіперпосилання.
	Edit	Редагування. Створює на формі поле для редагування текстового рядка.
	Memo	Поле, що відображає на формі поле для редагування текстових рядків. Зазвичай служить для створення редакторів і полів для виведення блоків даних.
	Button	Кнопка є найпоширенішим компонентом. Служить для створення в додатку різних прямокутних кнопок з текстовим написом.
	CheckBox	Незалежний перемикач, може приймати значення true/false, що змінюється при виборі мишею.
	RadioButton	Залежний перемикач, використовується для вибору тільки одного із декількох варіантів, для цього об'єднують такі варіанти у групу.
	ListBox	Список вибору містить список запропонованих варіантів (опцій) і дає змогу контролювати поточний вибір.
	ComboBox	Випадаючий список вибору, що представляє собою комбінацію компонентів Edit та ListBox
	Panel	Панель, цей компонент, як і GroupBox, використовується для об'єднання декількох компонентів.
	GroupBox	Контейнер групи компонентів, використовується для групування декількох зв'язаних між собою залежних компонентів.
	Image	Картинка. Компонент призначений для створення на формі невидимого контейнера, в який можна помістити один графічний файл.
	Bavel	Скіс, що створює об'ємні рамки для різних груп об'єктів. Служить тільки для обрамлення і не дозволяє виконувати програмну функцію.

	Image	Картинка. Компонент призначений для створення на формі невидимого контейнера, в який можна помістити один графічний файл.
	Bavel	Скіс, що створює об'ємні рамки для різних груп об'єктів. Служить тільки для обрамлення і не дозволяє виконувати програмну функцію.
	BitButton	Графічна кнопка - служить для створення на формі додатка кнопки із зображенням і написом.
	SpeedButton	Швидка кнопка - дозволяє створити на формі кнопку із зображенням без напису. Знайома нам по панелі швидких кнопок.
	MaskEdit	Форматоване введення - призначений для створення прямокутного поля введення даних в спеціально заданому форматі. Дозволяє перевірити коректність даних, що вводяться за допомогою маски.
	CheckListBox	Список з індикаторами. Індикатори дозволяють реалізувати множинний вибір елементів списку.
	DateEdit	Компонент являє собою рядок редагування для введення значень дат.
	DateTimePicker	Складальник дати і часу - створює в програмі інтерфейс для введення дати і часу.
	ProgressBar	Індикатор прогресу - створює на формі прямокутний індикатор для відображення процесу виконання довгих процедур в програмі (копіювання, пошуку та ін.).
	ListView	Вид списку - створює список елементів з відображенням в різних стилях (великі значки, дрібні значки, таблиця тощо.)
	TreeView	Вид дерева - дозволяє створювати в програмі ієрархічне деревовидне відображення даних.
	VirtualTree	Компонент призначений для відображення ієрархічного деревовидного представлення даних, але дані, які представляються з точки зору самого компонента - віртуальні. Компонент не знає нічого про їхній тип. Таким чином, компонент відповідальний за ієрархічну структуру даних і дії з вузлами.
	StringGrid	Рядкова таблиця - служить для створення таблиці (сітки), що складається з текстових рядків
	TabSet	TabSet - горизонтальні вкладки. Зазвичай використовується разом з Page для створення багатосторінкових вікон.
	Page	Сторінка – служить для створення сторінки, може містити TabSet

	StringGrid	Рядкова таблиця - служить для створення таблиці (сітки), що складається з текстових рядків
	TabSet	TabSet - горизонтальні вкладки. Зазвичай використовується разом з Page для створення багатосторінкових вікон.
	Page	Сторінка – служить для створення сторінки, може містити TabSet
	ToolBar	Панель інструментів - дозволяє створити на формі контейнер для розміщення швидких кнопок.
	MainMenu	Головне меню - призначене для створення головного меню програми.
	PopupMenu	Спливаюче меню - призначене для створення спливаючого меню деяких компонентів. Зазвичай за допомогою цього когось компонентами створюється контекстне меню.
	SplitterBox	Компонент призначений для перестроювання розмірів панелей користувачем. Компонент вставляється між двома панелями, вирівняними за допомогою їх властивості Align, і дозволяє користувачеві зрушувати кордон між цими панелями.
	Shape	Фігура - дозволяє малювати на формі прості геометричні фігури, такі як коло, квадрат, еліпс або прямокутник при зміні властивості Shape. Допускає зміну кольору фігур і їх штрихування при зміні властивостей Color (Колір) і Style (Стиль) групи Brush (Кисть).
	ColorBox	Компонент ColorBox є випадаючим списком кольорів. Використовується як простий і зручний вибір кольору користувачем.

4 Компоненти, що починаються з букв **DB** призначені для роботи з базою даних, використовуючи властивість **DataSet** – компонент-джерело даних.

Таблиця Компоненти вкладки **Дані**

Компонент	Назва компоненту	Пояснення
	DBLabel	Напис. Створює на формі текстову мітку або напис.
	DBEdit	DBEdit і DBMemo забезпечують перегляд і редагування полів запису бази даних
	DBMemo	
	DBCheckBox	Компонент являє собою майже повний аналог звичайного прапорця (компонент CheckBox) і призначений для відображення і редагування будь-яких даних, які можуть мати тільки два значення.

	DBEdit	DBEdit і DBMemo забезпечують перегляд і редагування полів запису бази даних
	DBMemo	
	DBCheckBox	Компонент являє собою майже повний аналог звичайного прапорця (компонент CheckBox) і призначений для відображення і редагування будь-яких даних, які можуть мати тільки два значення.
	DBComboBox	Компонент відображає поточне значення пов'язаного з ним поля набору даних в рядку редагування, при цьому значення поля має збігатися з одним з елементів списку, що розгортається. Поточне значення можна змінити на будь-яке фіксоване зі списку компонента. Функціонально компонент нічим не відрізняється від компонента ComboBox, що представляє собою комбінований список.
	DBListBox	Компонент є модифікацією для роботи з базами даних компонента TListBox, у якого він запозичив всі основні властивості і методи. Дозволяє редагувати пов'язане поле поточного запису набору даних шляхом вибору можливого значення зі списку. Список заповнюється за допомогою властивості Items.
	GridEx	Компонент є таблицею, комірки якої містять рядки символів. Забезпечує табличний спосіб відображення на екрані рядків даних.

Таблиця 10 – Компоненти вкладки **Розширені**

Компонент	Назва компоненту	Пояснення
	EditEx	Поле для введення текстового рядка
	Spr	Довідник
	IntSpr	Інтегрований довідник
	AnlCrd	Аналітична картотека
	IntPdr	Довідник підрозділів
	IntVo	Види оплат
	Graph	Граф
	Barcode	Штрих-код
	QRCode	QR-код

	Graph	Граф
	Barcode	Штрих-код
	QRCode	QR-код
	RichEditor	Поле для редагування багаторядкового тексту. Компонент RichEdit працює з текстом в форматі RTF.
	DocFlowDesigner	Готовий набір елементів для моделювання потоків даних
	DocFlowReader	Зчитує модель потоків даних, взаємодіє з компонентом DocFlowDesigner
	GridXML	Таблиця для баз даних, що призначена виключно для відображення та редагування пов'язаної з БД інформації.

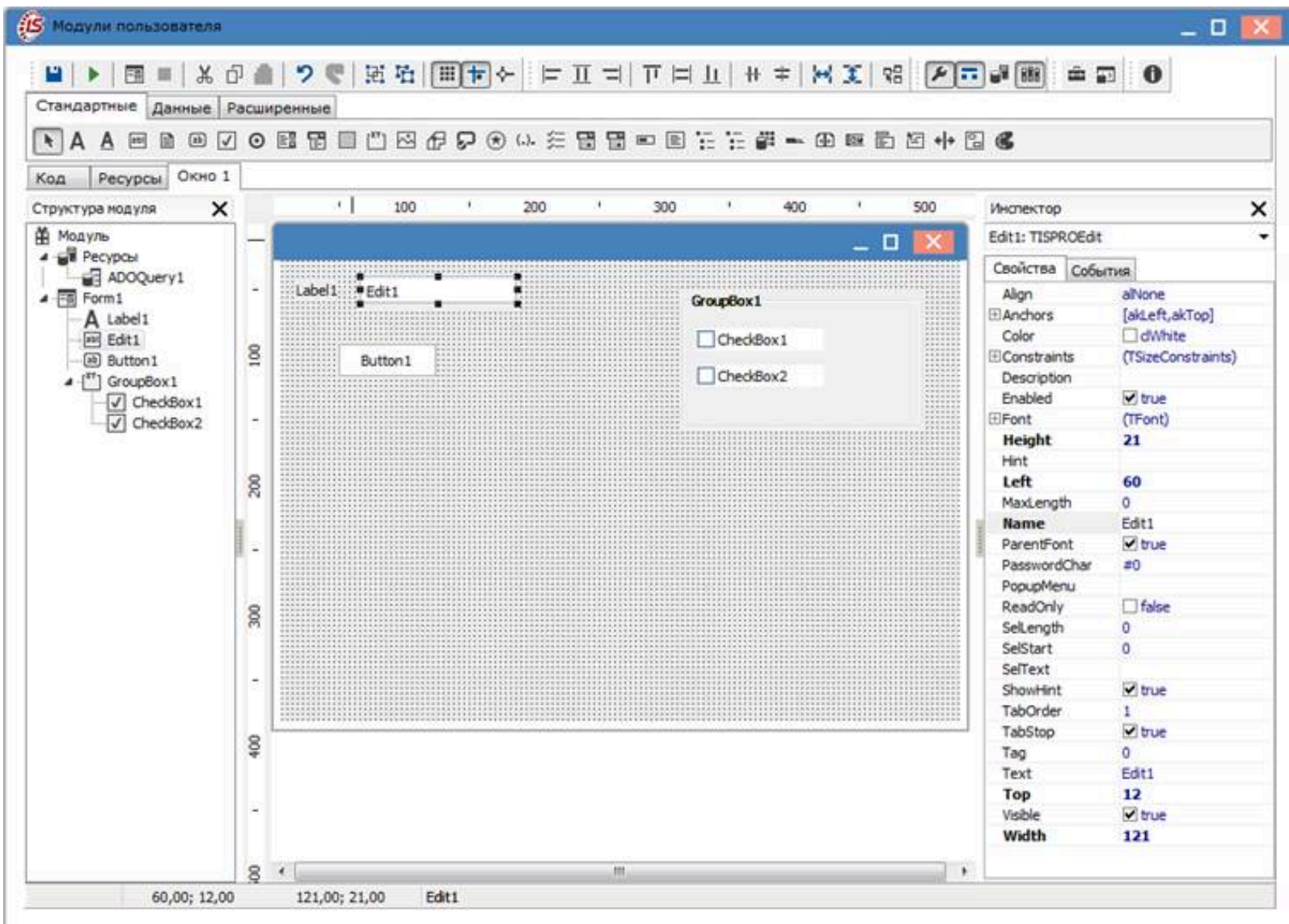


Рис. 7 – Дизайнер. Вкладка Вікно

1.1.1.1. Бібліотека

Бібліотека представляє собою збірник підпрограм, функцій або об'єктів, які використовуються при розробленні програм.

Бібліотека в Дизайнері викликається по кнопці-команді з головного меню панелі інструментів -



Бібліотека містить вкладки: **Поля БД, Змінні, Функції, Класи, Змінні ІСпро, Функції ІСпро.**

На вкладці **Поля БД** відображаються створені поля в таблицях БД. При необхідності відмічаються параметри: **Вставляти поле**, **Вставляти заголовок**, **Сортувати дерево даних**.

На вкладці **Змінні** відображаються змінні у вигляді ієрархічної структури. Основні розділи змінних **ІСпро** та **Системні**.

На вкладці **Функції у вигляді** ієрархічної структури відображаються функції, які використовуються при написанні програм. Основні розділи функцій: **Математичні**, **Агрегатні**, **Інші**, **Перетворення**, **Рядки**, **Дата і час**, **Системні**, **Форматування**.

Таблиця 11 – Функції Дизайнера

Функції	Опис
Математичні функції	
Abs(e: Extended): Extended	Модуль (абсолютне значення) цілого числа
ArcTan(X: Extended): Extended	Арктангенс
Cos(e: Extended): Extended	Косинус
Exp(X: Extended): Extended	Експонента
Frac(X: Extended): Extended	Повертає дробову частину числа
Int(e: Extended): Integer	Повертає цілу частину від числа
Ln(X: Extended): Extended	Натуральний логарифм
Pi: Extended	Число Пі
Round(e: Extended): Integer	Округлює число до вказаної кількості дрібних розрядів
Sin(e: Extended): Extended	Синус
Sqrt(e: Extended): Extended	Корінь квадратний
Tan(X: Extended): Extended	Тангенс
Trunc(e: Extended): Integer	Повертає цілочислену частину числа з плаваючою комою
Агрегатні функції	
AVG(Expr: Variant; Band: Variant = 0; Flags: Integer=0): Variant	Обчислює середнє значення числового стовпця
COUNT(Band: Variant = 0; Flags: Integer=0): Variant	Обчислює кількість записів в таблиці
MAX(Expr: Variant; Band: Variant = 0; Flags: Integer=0): Variant	Обчислює максимальне значення

COUNT(Band: Variant = 0; Flags: Integer=0): Variant	Обчислює кількість записів в таблиці
MAX(Expr: Variant; Band: Variant = 0; Flags: Integer=0): Variant	Обчислює максимальне значення
MIN(Expr: Variant; Band: Variant = 0; Flags: Integer=0): Variant	Обчислює мінімальне значення
SUM(Expr: Variant; Band: Variant = 0; Flags: Integer=0): Variant	Суму значень

Функції перетворення

BoolToStr(B: Boolean): string;	Перетворення логічного типу в рядок
DateTimeToStr(e: Extended): String;	Перетворення логічного типу в рядок
DateToStr(e: Extended): String;	Перетворення двох значень дати / часу в рядок
FloatToStr(e: Extended): String;	Перетворити число з плаваючою точкою в рядок
IntToStr(i: Integer): String;	Перетворити ціле число в рядок
StrToBool(const S: string): Boolean;	Перетворення рядка в логічний тип
StrToDate(s: String): Extended;	Перетворення рядка в дату
StrToDateTime (s: String): Extended;	Перетворення рядка в два значення дати / часу
StrToFloat(s: String): String;	Перетворення рядка в число з плаваючою точкою
StrToInt(s: String): Integer;	Перетворення рядка в ціле число
StrToInt64(s: String): Int64;	Перетворення рядка з цілим значенням в десятковій і шістнадцятковій числа
StrToTime(s: String): Extended;	Перетворення рядка в час
TimeToStr(e: Extended): String;	Перетворення часу в рядок
VarToStr(v: Variant): String;	Перетворення значення типу Variant в рядок.

StrToTime(s: String): Extended;	Перетворення рядка в час
TimeToStr(e: Extended): String;	Перетворення часу в рядок
VarToStr(v: Variant): String;	Перетворення значення типу Variant в рядок.
Рядкові функції	
Chr(i: Integer): Char	Функція дозволяє отримувати символ на основі його коду
CompareText(s, s1: String): Integer	Функція порівнює рядки A і B без урахування регістру. Якщо A = B, то повертається значення 0. Якщо A < B, то повертається значення -1. Якщо A > B, то повертається значення 1.
Copy(s: String; from, count: Integer): String	Функція дозволяє копіювати фрагмент деякої рядки з однієї змінної в іншу.
Delete(var s: String; from, count: Integer)	Вилучає з рядка зазначену кількість символів count, розпочинаючи з індексу
DeleteStr(var s: String; from, count: Integer)	Видаляє з початкового рядка S підрядок з позиції from довжиною count
Insert(s: String; var s2: String; pos: Integer)	Вставляє рядок str, розпочинаючи З індексу
Length(s: Variant): Integer	Повертає довжину рядка (без урахування символу завершення рядка)
Lowercase(s: String): String	Перетворює символи рядка на малі, тобто на нижній регістр
NameCase(s: String): String	Переклад першого символу у верхній регістр
Ord(ch: Char): Integer	Повертає номер заданого символу
Pos(substr, s: String): Integer	Повертає номер індексу, з якого розпочинається підрядок substr. Якщо рядок не містить substr, функція повертає 0
Set Length(var S: Variant; L: Integer)	Змінює довжину рядка на newlength, за потреби скорочуючи його
Trim(s: String): String	Вилучає початкові й кінцеві пробіли і повертає новий рядок без пробілів
Uppercase(s: String): String	Перетворює символи рядка на великі, тобто на верхній регістр

Set Length(var S: Variant; L: Integer)	Змінює довжину рядка на newlength, за потреби скорочуючи його
Trim(s: String): String	Вилучає початкові й кінцеві пробіли і повертає новий рядок без пробілів
Uppercase(s: String): String	Перетворює символи рядка на великі, тобто на верхній регістр

Функції дати та часу

Date: TDateTime	Поточна дата
DayOf(Date: TDateTime): Integer	День місяця
DayOfWeek(aDate: TDateTime): Integer	День тижня
DaysInMonth(nYear, nMonth: Integer): Integer	Днів у місяці
DecodeDate(Date: TDateTime; var Year, Month, Day: Word)	Переклад дати в рік, місяць і день
DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Word)	Переведення часу в години, хвилини і секунди
EncodeDate(Year, Month, Day: Word): TDateTime	Переклад року, місяця і дня в формат дати
EncodeTime (Hour, Min, Sec, MSec: Word): TDateTime	Переведення годинника, хвилин і секунд в формат часу
IncDay(Value: TDateTime; Increment: Integer): TDateTime	Додати день
IncHour(Value: TDateTime; Increment: Integer): TDateTime	Додати години
IncMinute(Value: TDateTime; Increment: Integer): TDateTime	Додати хвилини
IncMonth(Value: TDateTime; Increment: Integer): TDateTime	Додати місяць
IncYear (Value: TDateTime:	Додати рік

IncMinute(Value: TDateTime; Increment: Integer): TDateTime	Додати хвилини
IncMonth(Value: TDateTime; Increment: Integer): TDateTime	Додати місяць
IncYear (Value: TDateTime; Increment: Integer): TDateTime	Додати рік
IsLeapYear(Year: Word): Boolean	Високосний рік
MonthOf(Date: TDateTime): Integer	Місяць
Now: TDateTime	Поточні дата і час
Time: TDateTime	Поточний час
YearOf(Date: TDateTime): Integer	Рік

Функції форматування

Format(Fmt: String; Args: array): String	Форматування
FormatDateTime(Fmt: String; DateTime: TDateTime): String	Форматування дати і часу
FormatFloat(Fmt: String; Value: Extended): String	Форматування числа з плаваючою комою
FormatMaskText(EditMask: string; Value: string): string	Форматування рядка по масці

Системні функції

FileGetAttr(const FileName: string): Integer;	Повертає атрибути файлу
FileSearch(const FileName, DirectoryList: string): string;	Здійснює пошук файлу в одній або більше папках
FileSetAttr(const FileName: string; Attr: Integer): Integer;	Встановлює атрибути файлу
FindInDir(Path: string; Attributes: Integer; FileList:	Повертає список файлів в каталозі по шляху, включаючи маску файлів, із зазначенням атрибутів файлів.

FileSetAttr(const FileName: string; Attr: Integer): Integer;	Встановлює атрибути файлу
FindInDir(Path: string; Attributes: Integer; FileList: TStringList);	Повертає список файлів в каталозі по шляху, включаючи маску файлів, із зазначенням атрибутів файлів.
ScreenHeight: Integer;	
ScreenWidth: Integer;	
SelectDir (const Caption: string; const StarDir: String): String	
TextToShortCut(Text: String) : Integer;	

Інші функції

BlobToStr(ds: TISxDBDataset; fld : String): String	
Ceil(e: Extended): Integer	Функція повертає значення integer, що представляє найменше ціле, яке більше або дорівнює e.
CreateOleObject(Classname: String): Variant	Створення OLE-об'єкта
Dec(var i: Integer; decr: Integer =1)	Декремент
EctractFilePath(const FileName: string): string;	Витяг всіх частин повного імені файлу
Float2String(pVal: real) : String	Перетворення дійсного числа в рядок з урахуванням регіональних налаштувань роздільника
Floor(e: Extended): Integer	Функція використовується для округлення дробів до цілого в меншу сторону
GetBit(ds:TISxDBDataset; fld: String; num: integer):Boolean	Повертає значення біта з числа
GetBitInt(pVal: integer; num: integer): Boolean	
GetServerType: String;	
GetViewType: Integer;	
IIF(Expr: Boolean; TrueValue, FalseValue: Variant): Variant	IIF використовується для визначення, чи є вираз істиною або хибним. Якщо вираз істинний, IIF повертає одне значення; якщо

GetServerType: String;	
GetViewType: Integer;	
IIF(Expr: Boolean; TrueValue, FalseValue: Variant): Variant	IIF використовується для визначення, чи є вираз істиною або хибним. Якщо вираз істинний, IIF повертає одне значення; якщо воно помилкове, IIF повертає інше значення. Значення, що повертаються IIF, встановлює користувач
Inc(var i: Integer; incr: Integer =1)	Інкремент
InputBox(ACaption, APrompt, ADefault: string): string	Показує діалог, який просить користувача про введення тексту, із значенням за замовчуванням
InputQuery(ACaption, APrompt: string; var Value: string): Boolean	Відображає діалог, який просить користувача про введення тексту
MessageDlg(Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlg Buttons; HelpCtx: Longint): Integer	Відображає повідомлення, знак і обрані кнопки
ProcessMessages;	Даний метод зобов'язує додаток обробити повідомлення, які надійшли і очікують своєї черги
QuotedStr(str: String): String	Функція переводить рядок S в форму цитати. Функція додає символ одиночної лапки (') в початок і кінець рядка, а так само дублює кожен одиночний символ лапки всередині рядка.
RaiseException(Param: String)	Генерація виключення
Random: Extended	Генерація псевдовипадкового числа
Randomize	Ініціалізація генератора псевдовипадкових чисел
RoundTo(const AValue: Double; const ADigit: Integer): Double;	Округлити до зазначеної кількості цифр
SaveDirDialog(pPath: String; pTitle: String): String	
ShowCalculator(const AValue: Double): Double;	Функція виклику калькулятора системи
ShowMessage(Msg: Variant)	Виведення повідомлення

SaveDirDialog(pPath: String; pTitle: String): String	
ShowCalculator(const AValue: Double): Double;	Функція виклику калькулятора системи
ShowMessage(Msg: Variant)	Виведення повідомлення
ValidDate(cDate: String): Boolean	Перевірка валідності дати в рядку
ValidFloat(cFlt: String): Boolean	Перевірка валідності числа з плаваючою комою в рядку
ValidInt(clnt: String): Boolean	Перевірка валідності цілого в рядку
VarArrayCreate(Bounds: Array; Typ: Integer): Variant	Створення динамічного масиву
VarType(V: Variant): Integer	Функція служить для визначення кодового позначення типу даних змінної або властивості об'єкта.

1.1.1.2. Створювання програмних проектів у Дизайнері

Створювання програмних проектів у **Дизайнері** передбачає створення форми, розміщування на ній потрібних компонентів, визначення їхніх властивостей та створювання програмних відгуків на задані дії.

Послідовність роботи може бути такою:

1. запустити **Дизайнер**. При запуску автоматично створюється новий проект. Якщо **Дизайнер** вже працює для створення нового проекту слід з головного меню обрати кнопку  **Зберегти**;
2. винести на порожню форму з палітри компонентів усі потрібні компоненти для створення діалогового вікна програми;
3. задати на вкладці **Властивості Інспектора** об'єктів всі потрібні властивості для компонентів, розташованих на формі;
4. створити потрібні відгуки на події для розташованих на формі компонентів. Це можна зробити за допомогою вкладки **Події Інспектора** об'єктів чи то безпосередньо з форми, натиснувши двічі по компонентів. Не можна самостійно прописувати заголовки відгуків чи вилучати автоматично створені;

5. у вікні редактора коду заповнити відгуки на події кодом, а також написати текст усіх потрібних програм;
6. зберегти створений проект за допомогою кнопки-команди головного меню  **Зберегти**;
7. запустити проект на виконання можна натиснувши кнопку  **Виконати** на панелі інструментів;
8. якщо помилок немає, з'явиться форма без координатної сітки. Якщо ж помилки є, то рядок з помилкою висвічуватиметься червоним кольором, а повідомлення компілятора, розташоване під вікном редактора коду, повідомить, якої саме помилки було припущено.
9. провести тестові розрахунки, для чого виконати відповідні дії (вводити дані, натискати потрібні кнопки тощо). Якщо при виконанні програми були виявлені алгоритмічні помилки їх, зрозуміло, слід виправити. Для довчасного припинення виконання програми, приміром, за її зависання, слід виконати команду  **Зупинити**;
10. доволі часто на практиці при налагодженні програми виникає потреба продивитися проміжкові результати обчислень, приміром за обчислень у циклах сум доданків. Для цього існують спеціальні "гарячі" кнопки покрокового виконання програми **Відладка**  (із заходом до підпрограм) і **Виконати до поточної позиції**  (те саме, але без заходу до підпрограм). Для переглядання результатів покрокового виконання програми слід відкрити відповідне віконце командою **Обчислити** . Щоб покрокове виконання програми здійснювати, розпочинаючи з певної команди програми, цю команду слід позначити так званою **Точкою зупинки**  ;
11. після налагодження й тестового виконання програми здійснити потрібні обчислення і вийти з **Дизайнера**. Сеанс завершено.
12. Рекомендується взяти за правило: перед кожним запуском програми обов'язково її зберігати.

1.1.1.3. Класи

1.1.1.3.1. Клас TADOConnection

Клас TADOConnection забезпечує з'єднання з даними, доступ до яких реалізується через ADO-об'єкти.

ADO-об'єкти дозволяють працювати з різними сховищами даних, які можуть і не бути SQL-операторами. Об'єкти типу TADOConnection використовують для доступу до даних OLE DB провайдери.

Компоненти TADOCommand і TADODataSet зв'язуються з джерелом даних за допомогою об'єкта TADOConnection, вказуючи посилання на нього як значення властивості Connection.

Для ідентифікації з'єднання необхідно визначити значення властивостіConnectionString (рядок з'єднання) компонента TADOConnection, яке може ґрунтуватися на вказівці:

- datalink-файлу;
- рядки з'єднання.

Якщо в якості значення властивостіConnectionString вказано ім'я UDL-файлу, то налаштування з'єднання можна виконувати автономно від програми (наприклад, вказуючи ім'я бази даних Microsoft SQL Server на поточному ПК).

Таблиця – Властивості класу TADOConnection = class(TComponent)

Властивості (property)

Attributes: TXactAttributes

CommandTimeout: Integer

Connected: Boolean

ConnectionString: String

ConnectionTimeout: Integer

ConnectOptions: TConnectOption

CursorLocation: TCursorLocation

DefaultDatabase: String

IsolationLevel: TIsolationLevel

KeepConnection: Boolean

LoginPrompt: Boolean

Mode: TConnectMode

Name: String

Provider: String

Tag: Integer

1.1.1.3.2. Клас TADOQuery

Компонент TADOQuery надає розробникам можливість для отримання даних з однієї або декількох таблиць з бази даних ADO за допомогою SQL.

Ці оператори SQL можуть бути або DDL (Data Definition Language) оператори, такі як CREATE TABLE, ALTER INDEX, і так далі, або ж вони можуть бути DML (Data Manipulation Language) заяви, такі як SELECT, UPDATE і DELETE. Найбільш спільну заяву, однак, є SELECT, заяву, яке виробляє вид, подібний до тих, які доступні за допомогою компонента Table.

Таблиця – Властивості та процедури класу TADOQuery = class(TCustomADODataSet)

Властивості (property)

Active: Boolean

AutoCalcFields: Boolean

CacheSize: Integer

CommandTimeout: Integer

Connection: TADOConnection

ConnectionString: String

CursorLocation: TCursorLocation

CursorType: TCursorType

DataSource: TDataSource

EnableBCD: Boolean

ExecuteOptions: TExecuteOptions

Filter: String

Filtered: Boolean

LockType: TADOLockType

MarshalOptions: TMarshalOption

MaxRecords: Integer

Name: String

ParamCheck: Boolean

Parameters: TParameters

Prepared: Boolean

SQL: TWideString

Tag: Integer

Parameters: TParameters

Prepared: Boolean

SQL: TWideString

Tag: Integer

Процедури (procedure)

ExecSQL

1.1.1.3.3. Клас TADOStoredProc

Компонент **TADOStoredProc** призначений для виконання збережених процедур бази даних.

Компонент **TADOStoredProc** може також вважатися приватною версією більш загального компонента **TADODataSet**, властивість **CommandType** яка має значення **cmdStoredProc**.

Компонент **TADOStoredProc** має властивість **ProcedureName** (WideString), служить для вказівки збереженої процедури для запуску. Параметри процедури визначаються властивістю **Parameters** і задаються так само, як і параметри **TADOQuery**. Для виконання процедури використовується метод **ExecProc**.

Таблиця – Властивості та процедури класу **TADOStoredProc = class(TCustomADODataSet)**

Властивості (property)

Active: Boolean

AutoCalcFields: Boolean

CacheSize: Integer

CommandTimeout: Integer

ConnectionString: String

CursorLocation: TCursorLocation

CursorType: TCursorType

DataSource: TDataSource

EnableBCD: Boolean

ExecuteOptions: TExecuteOptions

Filter: String

Filtered: Boolean

LockType: TADOLockType

MarshalOptions: TMarshalOption

MaxRecords: Integer

Filtered: Boolean

LockType: TADOLockType

MarshalOptions: TMarshalOption

MaxRecords: Integer

Name: String

Params: TParams

Prepared: Boolean

ProcedureName: String

Tag: Integer

Процедури (procedure)

ExecSQL

1.1.1.3.4. Клас TADOTable

Компонент **TADOTable** використовується в тих випадках, коли необхідно працювати з однією таблицею в базі даних. Основні властивості і методи компонента **TADOTable** аналогічні властивостям компонента **TADODataset**. Крім того, цей компонент по своїй функціональності аналогічний компоненту **TTable**.

Властивістю **TableName** задаєть ім'я таблиці БД. Зв'язок компонентів **TADOTable**, які працюють з різними таблицями, одна з яких головна, а інша - допоміжна, здійснюється за допомогою властивостей **MasterSource** і **MasterFields**. **MasterFields** властивість застосовується для вказівки одного і більше полів з головної таблиці, для реалізації відносини рядків «один до багатьох».

MasterSource (TDataSource) - містить посилання на головну таблицю при відношенні типу «один до багатьох». Якщо властивість **TableDirect** має значення **True**, здійснюється прямий доступ до таблиці. В іншому випадку компонент генерує відповідний запит. Властивість **ReadOnly** дозволяє включити або відключити для таблиці режим «тільки для читання».

Таблиця 15 – Властивості та процедури класу **TADOTable = class(TCustomADODataset)**

Властивості (property)

Active: Boolean

AutoCalcFields: Boolean

CacheSize: Integer

CommandTimeout: Integer

ConnectionString: String

CursorLocation: TCursorLocation

CacheSize: Integer
CommandTimeout: Integer
ConnectionString: String
CursorLocation: TCursorLocation
CursorType: TCursorType
EnableBCD: Boolean
ExecuteOptions: TExecuteOptions
Filter: String
Filtered: Boolean
IndexFieldNames: String
IndexName: String
LockType: TADOLockType
MarshalOptions: TMarshalOption
MasterFields: String
MasterSource: TDataSource
MaxRecords: Integer
Name: String
ReadOnly: Boolean
TableDirect: Boolean
TableName: String
Tag: Integer

1.1.1.3.5. Клас TApplication

Клас **Application** призначений для опису структури і властивостей клієнтського додатка, що безпосередньо запускається користувачем.

Таблиця 16 – Властивості та процедури класу **TApplication = class(TComponent)**

Властивості (property)
Name: String
Tag: Integer
Процедури (procedure)
Minimize

Name: String

Tag: Integer

Процедури (procedure)

Minimize

ProcessMessages

Restore

ExeName: String

1.1.1.3.6. Клас TAutoIncField

Таблиця – Властивості та процедури класу **TAutoIncField = class(TIntegerField)**

Властивості (property)

Alignment: TAlignment

AutoGenerateValue: TAutoRefreshFlag

ConstraintErrorMessage: String

CustomConstraint: String

DefaultExpression: String

DisplayFormat: String

DisplayLabel: String

DisplayWidth: Integer

EditFormat: String

FieldKind: TFieldKind

FieldName: String

HasConstraints: Boolean

ImportedConstraint: String

Index: Integer

KeyFields: String

LookupCache: Boolean

LookupDataSet: TDataSet

LookupKeyFields: String

LookupResultField: String

MaxValue: Integer

LookupDataSet: TDataSet
LookupKeyFields: String
LookupResultField: String
MaxValue: Integer
MinValue: Integer
Name: String
Origin: String
ProviderFlags: TProviderFlags
ReadOnly: Boolean
Required: Boolean
Tag: Integer
Visible: Boolean

1.1.1.3.7. Клас TBCDField

Таблиця – Властивості та процедури класу **TBCDField = class(TNumericField)**

Властивості (property)

Alignment: TAlignment
AutoGenerateValue: TAutoRefreshFlag
ConstraintErrorMessage: String
currency: Boolean
CustomConstraint: String
DefaultExpression: String
DisplayFormat: String
DisplayLabel: String
DisplayWidth: Integer
EditFormat: String
FieldKind: TFieldKind
FieldName: String
HasConstraints: Boolean
ImportedConstraint: String

FieldName: String
HasConstraints: Boolean
ImportedConstraint: String
Index: Integer
KeyFields: String
LookupCache: Boolean
LookupDataSet: TDataSet
LookupKeyFields: String
LookupResultField: String
MaxValue: Integer
MinValue: Integer
Name: String
Origin: String
Precision: Integer
ProviderFlags: TProviderFlags
ReadOnly: Boolean
Required: Boolean
Size: Integer
Tag: Integer
Visible: Boolean

1.1.1.3.8. Клас TBevel

Компонент **Bevel** (кромка) призначений для виділення і розмежування областей на формі. Може мати вигляд прямокутника або лінії. Об'єкт надає менше можливостей у порівнянні з компонентом **Panel** (класу **Tpanel**), але він не займає ресурсів. Зовнішній вигляд вказується за допомогою властивостей **Shape** і **Style**.

Геометрична форма компонента визначається його властивостями.

Таблиця – Властивості та процедури класу **TBevel**= **class(TGraphicControl)**

Властивості (property)

Align: TAlign

AlignWithMargins: Boolean

Anchors: TAnchors

Constraints: TSizeConstraint

Cursor: Integer

Height: Integer

HelpContext: Integer

HelpKeyword: String

HelpType: THelpType

Hint: String

Left: Integer

Margins: TMargins

Name: String

ParentShowHint: Boolean

Shape: TBavelShape

ShowHint: Boolean

Style: TBavelStyle

Tag: Integer

Top: Integer

Visible: Boolean

Width: Integer

1.1.1.3.9. Клас TBinaryField

Таблиця 20 – Властивості та процедури класу **TBinaryField = class(TField)****Властивості (property)**

Alignment: TAlignment

AutoGenrateValue: TAutoRefreshFlag

ConstraintErrorMessage: String

CustomConstraint: String

DefaultErrorMessage: String

AutoGenrateValue: TAutoRefreshFlag
ConstraintErrorMessage: String
CustomConstraint: String
DefaultExpression: String
DisplayLabel: String
DisplayWidth: Integer
FieldKind: TFieldKind
FieldName: String
HasConstraints: Boolean
ImportedConstraint: String
Index: Integer
KeyFields: String
LookupCache: Boolean
LookupDataSet: TDataSet
LookupKeyFields: String
LookupResultField: String
Name: String
Origin: String
ProviderFlags: TProviderFlags
ReadOnly: Boolean
Required: Boolean
Size: Integer
Tag: Integer
Visible: Boolean

1.1.1.3.10. Клас TBitBtn

Піктографічна кнопка **TBitBtn** представляє собою різновид стандартної кнопки **TButton**. Її відмінна риса - властивість **Glyph**, за допомогою якого визначається растрове зображення на поверхні кнопки.

Таблиця 21 – Властивості та процедури класу **TBitBtn = class(TButton)**

Властивості (property)

Active: TBasicAction

Align: TAlign

AlignWithMargins: Boolean

Anchors: TAnchors

BiDiMode: TBiDiMode

Cancel: Boolean

Caption: String

Constraints: TSizeConstraint

Cursor: Integer

Default: Boolean

DragCursor: Integer

DragKind: TDragKind

DragMode: TDragMode

Enabled: Boolean

Font: TFont

Glyph: TBitmap

Height: Integer

HelpContext: Integer

HelpKeyword: String

HelpType: THelpType

Hint: String

Kind: TBitBtnKind

Layout: TButtonLayout

Left: Integer

Kind: TBitBtnKind
Layout: TButtonLayout
Left: Integer
Margin: Integer
Margins: TMargins
ModalResult: Integer
Name: String
NumGlyph: Integer
ParentBiDiMode: Boolean
ParentFont: Boolean
ParentShowHint: Boolean
PopupMenu: TPopupMenu
ShowHint: Boolean
Spacing: Integer
Style: TButtonStyle
TabOrder: Integer
TabStop: Boolean
Tag: Integer
Top: Integer
Visible: Boolean
Width: Integer
WordWrap: Boolean

1.1.1.3.11. Клас TBitmap

Клас **TBitmap** є основою растрової графіки, він визначає клас зображень або бітових карт, де зображення задається кольорами всіх своїх пікселів. Це єдиний стандартний клас, який дозволяє редагувати зображення: в програмі - канва бітової карти доступна через властивість **Canvas** (інші нащадки **TGraphic** канви не мають).

Таблиця – Властивості та процедури класу **TBitmap = class(TGraphic)**

Властивості (property)

Canvas: TCanvas

1.1.1.3.12. Клас TBlobField

Клас **TBlobField** використовується для роботи з полями бази даних, представлених у вигляді великої кількості бінарних даних (**binary large object-BLOB**).

Таблиця – Властивості та процедури класу **TBlobField = class (TField)**

Властивості (property)

Alignment: TAlignment

AutoGenrateValue: TAutoRefreshFlag

BlobType: TBlobType

ConstraintErrorMessage: String

CustomConstraint: String

DefaultExpression: String

DisplayLabel: String

DisplayWidth: Integer

FieldKind: TFieldKind

FieldName: String

GraphicHeader: Boolean

HasConstraints: Boolean

ImportedConstraint: String

Index: Integer

KeyFields: String

LookupCache: Boolean

LookupDataSet: TDataSet

LookupKeyFields: String

LookupResultField: String

Name: String

Origin: String

ParentField: TField

LookupResultField: String

Name: String

Origin: String

ProviderFlags: TProviderFlags

ReadOnly: Boolean

Required: Boolean

Size: Integer

Tag: Integer

Visible: Boolean

Процедура (procedure)

LoadFromFile(const FileName: String)

LoadFromStream(Stream: TStream)

SaveToFile(const FileName: String)

SaveToStream(Stream: TStream)

1.1.1.3.13. Клас TBooleanField

BooleanField - клас полів сутності для даних булевого типу.

Таблиця – Властивості та процедури класу **BooleanField = class (TField)**

Властивості (property)

Alignment: TAlignment

AutoGenerateValue: TAutoRefreshFlag

BlobType: TBlobType

ConstraintErrorMessage: String

CustomConstraint: String

DefaultExpression: String

DisplayLabel: String

DisplayWidth: Integer

FieldKind: TFieldKind

FieldName: String

GraphicHeader: Boolean

FieldKind: TFieldKind

FieldName: String

GraphicHeader: Boolean

HasConstraints: Boolean

ImportedConstraint: String

Index: Integer

KeyFields: String

LookupCache: Boolean

LookupDataSet: TDataSet

LookupKeyFields: String

LookupResultField: String

Name: String

Origin: String

ProviderFlags: TProviderFlags

ReadOnly: Boolean

Required: Boolean

Size: Integer

Tag: Integer

Visible: Boolean

Процедура (procedure)

LoadFromFile(const FileName: String)

LoadFromStream(Stream: TStream)

SaveToFile(const FileName: String)

SaveToStream(Stream: TStream)

1.1.2. Дизайнер вікон. Компоненти

Для створення нового користувацького вікна необхідно в меню модуля **Конструктор** у вкладці **Вікна** обрати пункт **Створення** (клавіша **Ins**). У вікні в полі **Номер** ввести унікальний номер вікна, а в полі **Найменування** - назву призначеного для користувача вікна. По кнопці **ОК** створюється новий запис в реєстрі. По кнопці **Дизайнер** відкриється дизайнер, в якому можна редагувати вигляд вікна, логіку обробки подій, які відображаються користувачу дані.

Це інструмент для створення користувацьких вікон. Дизайнер за своїм виглядом дуже нагадує середу розробки звітів **Fast Report**. Робочий простір розділено на три області: **Код**, **Ресурси**, **Вікно**.

В дизайнері вікон по кнопці **Закоментувати / Розкоментувати текст** (комбінація клавіш **Ctrl + K**) можна залишити / зняти коментарі. При створенні призначених для користувача форм по клавіші **F3** викликаються елементи програм.

На вкладці **Код** можна писати свою логіку відображення, обробки, збереження даних. Дизайнер підтримує мови **PascalScript**, **Jscript**, **BasicScript**, **C ++ Script**. Тут так само підтримується всі функції, поля, реєстри і довідники **Комплексу**, які доступні в звітах.

На вкладці **Ресурси** проводиться робота з запитамі, читання даних з **Комплексу**, робота з тимчасовими таблицями, файлами DBF. На робочу область програми можна виносити такі елементи програми як **ImageList** (використовується для відображення іконок в компоненті **ToolBar**), запит ODBC, Тимчасова таблиця, Таблиця DBF.

На вкладці **Вікно** можна створювати свій дизайн вікна, виводити на форму такі стандартні елементи як **Label**, **Edit**, **Button**, **Memo**, **RadioButton**, **ListView** і інші. На закладці **Дані** виведені спеціальні компоненти, у властивостях яких, можна вказати **Dataset** і поле, з якого отримувати дані. На вкладці **Розширені** містяться елементи для роботи з довідниками **Комплексу**. Вони аналогічні по своїй роботі з довідниками, які вбудовані в **FastReport**.

У дизайнера користувацьких вікон доступний компонент **Граф (TISPROGraph)**. Компонент може відображати схеми бізнес-процесів у вигляді документів підсистем і їх напрямки руху або довільні схеми роботи. Компонент має функції, за допомогою яких користувачі можуть самостійно, програмно, створювати графи, наповнювати та керувати ними.

1.1.2.1. Компонент TISPROGraph

Розташування: модуль **Конструктор**, вкладка **Розширені**.

1. Подвійний клік відкриває редактор графа.

У редакторі реалізовані можливості:

- завантаження з файлу;
- об'єднання з файлу;
- збереження у файл;
- додавання вузла і зв'язку.

Додані константи атрибутів файлів. Реалізовані функції:

- **FileSearch** - Здійснює пошук файлу в одній або більше папках;
- **FindInDir** - Повертає список файлів в каталозі по шляху, включаючи маску файлів, із зазначенням атрибутів файлів.
- **FileGetAttr** - Повертає атрибути файлу;
- **FileSetAttr** - Встановлює атрибути файлу.

2. У редакторі подвійний клік на елементі відкриває вікно властивостей вузла або зв'язку.

Вікно властивостей вузла:

- Тема, його розташування і шрифт;
- Вид (Квадрат, коло, трикутник ...);
- Розміри;
- Кольори фону і кордонів;
- Стилi заливки і рамки;
- Розширені можливості пошуку (код та ідентифікатор документа);
- Вибирати зображення (іконку) з файлу;
- Налаштовувати розташування іконки;
- Вибирати зображення з бібліотеки **Комплексу**.

Вікно властивостей зв'язку:

- Підпис і її розташування;
- Види і розміри країв зв'язків;
- Стилi зв'язку;
- Кольори ліній і заливки країв.

Функції для програмного доступу:

- **Завантажити з файлу:** function LoadFromFile (FileName: String): Integer;
- **Об'єднати з файлом:** function Integerfunction MergeFromFile (FileName: String): Integer;
- **Зберегти у файл:** function SaveToFile (FileName: String): Integer;

- **Отримати ID натиснутого об'єкта:** function GetClickedObject: integer;
- **Отримати ID натиснутого вузла:** function GetClickedNode: integer;
- **Отримати ID натиснутою зв'язку:** function GetClickedLink: integer;
- **Додати вузол:** function AddNode (X, Y: Integer; Cap: String; ImageIndex: Integer): Integer;
- **Присвоїти реквізити документа:** procedure SetNodeDoc (NodeID, DocCd, DocRcd: Integer; Cap: String; ImageIndex: Integer): Integer;
- **Отримати ID по реквізитами документа:** function GetDocID (DocCd, DocRcd: Integer): Integer;
- **Додати зв'язок:** function AddLink (SourceID, TargetID: Integer): Integer;
- **Отримати ID зв'язку з ID пов'язаних вузлів:** function GetLinkID (SourceID, TargetID: Integer): Integer;
- **Видалити елемент графа:** procedure DeleteGraphObj (ID: Integer);
- **Отримати код документа по ID вузла:** function GetDocCd (ID: Integer): Integer;
- **Отримати ID документа по ID вузла:** function GetDocRcd (ID: Integer): Integer;
- **Встановити розмір шрифту вузла:** procedure NodeFontSize (ID, Size: Integer);
- **Встановити колір шрифту вузла:** procedure NodeFontColor (ID: Integer; color: TColor);
- **Встановити стиль вузла:** procedure NodeBodyStyle (ID: Integer; style: TBrushStyle);
- **Встановити колір вузла:** procedure NodeBodyColor (ID: Integer; color: TColor);
- **Встановити стиль рамки вузла:** procedure NodeBorderStyle (ID: Integer; style: TPenStyle);
- **Встановити колір рамки вузла:** procedure NodeBorderColor (ID: Integer; color: TColor);
- **Встановити товщину рамки вузла:** procedure NodeBorderWidth (ID: Integer; value: Integer);
- **Встановити ширину вузла:** procedure NodeWidth (ID: Integer; value: Integer);
- **Встановити висоту вузла:** procedure NodeHeight (ID: Integer; value: Integer);
- **Встановити форму вузла:** procedure NodeShape (ID: Integer; value: Integer);
- **Встановити розмір шрифту зв'язку:** procedure LinkFontSize (ID, Size: Integer);
- **Встановити колір шрифту зв'язку:** procedure LinkFontColor (ID: Integer; color: TColor);
- **Встановити стиль початку зв'язку:** procedure LinkBeginStyle (ID, style: Integer);
- **Встановити розмір початку зв'язку:** procedure LinkBeginSize (ID, Size: Integer);
- **Встановити стиль закінчення зв'язку:** procedure LinkEndStyle (ID, style: Integer);
- **Встановити розмір закінчення зв'язку:** procedure LinkEndSize (ID, Size: Integer);
- **Встановити товщину зв'язку:** procedure LinkWidth (ID, Size: Integer);
- **Встановити стиль зв'язку:** procedure LinkStyle (ID: Integer; style: TPenStyle);
- **Встановити текст зв'язку:** procedure LinkText (ID: Integer; Text: String);
- **Встановити колір зв'язку:** procedure LinkLineColor (ID: Integer; color: TColor);
- **Встановити колір заливки зв'язку:** procedure LinkFillColor (ID: Integer; color: TColor);
- **Встановити колір шрифту колонки по імені поля колонки:** SetFieldFontColor;
- **Встановити колір фону колонки по імені поля колонки:** SetFieldBgColor;
- **Встановити шрифту і фону для рядка:** SetRowBgFntColor.

Події:

- **Подвійне натискання на вузлі:** OnNodeDbClick;
- **Подвійне натискання на зв'язку:** OnLinkDbClick;
- **Натискання на вузлі:** OnNodeClick;
- **Натискання на зв'язку:** OnLinkClick.

1.1.2.2. Компонент VirtualTree

Компонент **VirtualTree** призначений для відображення деревовидних даних на користувальницької формі. У дизайнера вікон розміщений на вкладці **За умовчанням**.

Функції***Функції створення структури дерева***

- **procedure AddNode (ID: Int64; ParentID: Int64; Text: String)**
 - Додає гілку з ім'ям **Text** в дерево, **ID** якій повинно бути унікальним.
 - **ParentID** - ID гілки, в яку буде додаватися нова гілка.
- **procedure AddColumn (Name: String)**
 - Додає у дерево нову колонку з ім'ям **Name**. Текст в гілках нової колонки встановлюється з першої колонки відповідної гілки.

Функції заповнення дерева

- **procedure SetText (NodeID: Int64; ColumnIndex: Int64; Text: String)**
 - Встановлює текст **Text** в осередку гілки з номером **NodeID** і колонки з номером **ColumnIndex** (нумерація починається з 0).
- **procedure SetColumnName (ColumnIndex: Integer; Name: String)**
 - Встановлює ім'я **Name** колонки з номером **ColumnIndex**.
- **procedure SetNodeCheck (ID: Int 64; Checked: Boolean)**
 - Встановлює або знімає позначку з гілки з номером **ID**.

Функції навігації по дереву

- **procedure SetFocusedNode (ID: Int64)**
 - Встановлює курсор на гілці з номером **ID**, при цьому фокус колонки не змінюється.
- **procedure SetFocusedColumn (ColumnIndex: Integer)**
 - Встановлює курсор на колонці з номером **ColumnIndex**, при цьому фокус гілки не змінюється.

- **procedure ExpandNode (ID: Int64);**
 - Розкриває гілку з номером ID;
- **procedure CollapsNode (ID: Int64);**
 - Згортає гілку з номером ID;
- **procedure ExpandAll ();**
 - Розкриває всі гілки;
- **procedure CollapsAll ();**
 - Згортає всі гілки.

Функції зворотного зв'язку з деревом

- **function GetText (NodeID: Int 64; ColumnIndex: Int 64): String**
 - Повертає текст в осередку, яка розміщена в гілці з номером **NodeID** і колонці з номером **ColumnIndex**.
- **function GetFocusedText(): String**
 - Повертає текст у виділеній комірці.
- **function GetSelectedNodeID (): Int64**
 - Повертає номер гілки, на якій знаходиться курсор.
- **function GetSelectedColumnIndex (): Int64**
 - Повертає номер колонки, на якій знаходиться курсор.
- **function GetNodeCheck (ID: Int64): Boolean**
 - Повертає true або false, що відображають відзначена чи гілка дерева.
- **function GetNodeParentID (ID: Int 64): Int 64**
 - Повертає ID батька гілки з номером ID.

Функції циклічного перебору структури дерева

- **function GetFirstNodeID(): Int64**
 - Повертає ID першої гілки.
- **function GetNextNodeID (ID: Int 64): Int 64**
 - Повертає ID наступної за переданої в параметрі функції гілки.
- **function GetNodeCount (): Int64**
 - Повертає кількість гілок в дереві.
- **function GetColumnCount (): Int64**
 - Повертає кількість колонок в дереві.

Код: програми:

- Заповнення дерева:
 - *procedure Button1OnClick (Sender: TISPROComponent);*
 - *var*
 - *i: Int64;*
 - *begin*
 - *VirtualTree1.AddNode (1,0, '1');*
 - *VirtualTree1.AddNode (2,1, '11');*
 - *VirtualTree1.AddNode (11,0, '2');*
 - *VirtualTree1.AddNode (12,2, '111');*
 - *VirtualTree1.AddNode (17,2, '112');*
 - *VirtualTree1.AddNode (18,11, '21');*
 - *VirtualTree1.AddColumn ('col');*
 - *end;*
- Встановити тест в певній клітинці:
 - *procedure Button2OnClick (Sender: TISPROComponent);*
 - *begin*
 - *VirtualTree1.SetText (2,1, 'dcdvdv');*
 - *end;*
- Встановити фокус в вибраній комірці:
 - *procedure Button5OnClick (Sender: TISPROComponent);*
 - *begin;*
 - *VirtualTree1.SetFocusedColumn (1);*
 - *VirtualTree1.SetFocusedNode (11);*
 - *end.*
- Отримати текст комірки:
 - *procedure Button6OnClick (Sender: TISPROComponent);*
 - *begin;*
 - *ShowMessage (VirtualTree1.GetText (2, 1));*
 - *end.*

1.1.2.3. Компонент XMLGrid

Компонент «**TISPROGrid XML**» призначений для відображення даних в табличній формі з використанням XML - реєстру на користувацькій формі. В дизайнері вікон розміщений на вкладці **Розширені**.

Параметри компонента:

- **XMLFileDescription** – ім'я файлу XML -реєстра:
 - **MdID** - код модуля;
 - **RstID** - код реєстру;
 - **QueryID** - код запиту.

Значення з імені файлу **XML** - реєстра.

Наприклад: U055_1_001.xml - MdID = 55, RstID = 1, QueryID = 1

- **GridID** - ідентифікатор реєстру;
- **XMLBaseDescription** - опис таблиці:
 - **BaseSynonym** - Ім'я таблиці. Синонім для таблиці в реєстрі;
 - **UniqueKeys** - Список полів, що становлять унікальний ключ записи. Обов'язковий параметр для позиціонування на реєстрі;
 - **MarkAbsField** - Ім'я поля, значення якого буде використовуватися для оцінки записів. Обов'язковий параметр для роботи із зазначеними записами;
 - **Mrk** - Код маркера реєстру (повинен бути більше 20 000).

Події:

- **OnClick** - приходить при будь-якій зміні фокуса в реєстрі, тобто при зміні осередку, в т.ч. в межах одного рядка;
- **OnShowRecord** - викликається при відображенні запису.
- **OnModFieldBegin** - викликається на початку редагування поля.
- **OnModifyField** - викликається після завершення редагування поля.
- **OnNewRecord** - викликається при створенні нового запису в реєстрі.
- **OnInsertRecord** - викликається при вставці запису до реєстру.
- **_ShowRecord** - компонент **Таблиця** видає це повідомлення перед завантаженням записи у внутрішній буфер. Оброблювач цього повідомлення повинен заповнити всі обчислюваності поля, що входять до запису таблиці. Тобто сформувати запис, який потім буде відображатися.
- **_BeforeRecord** - це повідомлення надсилається при позиціонуванні на запис, після того, як поточна запис на базі встановлена. Оброблювач цієї події повинен обробляти зв'язку таблиці з іншими елементами форми: оновлювати праву частину комбореєстра, оновлювати поля підвалу, оновлювати пов'язані таблиці і т.п. Т.е. Тобто ця подія - сигнал навігації по таблиці.

Функції:

Функції ініціалізації і інтерфейсу:• **function Init: integer**

Ініціалізація роботи з реєстром. Функція ініціалізує механізм і викликає побудову / заповнення реєстру згідно з видом, зазначеним в параметрах. Перед викликом обов'язково повинні бути заповнені параметри: **GridID, MdID, RstID, QueryID**.

• **function Refresh: integer**

- Оновлює дані в реєстрі. Оновлення здійснюється з повторним запитом параметрів у користувача.

• **function RefreshNoAsk: integer**

- Оновлює дані в реєстрі. Оновлення здійснюється з попередніми значеннями параметрів.

• **function RefreshNoClear: integer**

- Оновлює дані в реєстрі без очищення реєстру. Можна застосовувати, наприклад, для оновлення не лише реєстру, а однієї або декількох записів, попередньо їх видаливши.

• **function SelectView: integer**

- Виводить вікно вибору / встановлення виду реєстру. (Відповідає натисканню комбінації клавіш **Alt + 0**)

• **function RefreshRst: integer**

- Оновлює дані в реєстрі. Оновлення здійснюється з повторним запитом параметрів і очищенням маркерів (відповідає натисканню комбінації клавіш **Ctrl + F2**).

• **function ShowFindWindow: integer**

- Показує вікно пошуку. Відповідає натисканню клавіші **F7**.

• **function ShowFilterWindow: integer**

- Показує вікно фільтра. Відповідає натисканню клавіші **F11**.

• **function ShowFilterWindowCurrent: integer**

- Показує вікно фільтра з заповненням значення за поточним полем. Відповідає натисканню комбінації клавіш **Alt + F11**.

Функції роботи з таблицею реєстру:• **function SetBegin: integer**

- Встановити курсор на перший запис.

• **function GetNext: integer**

- Встановити курсор на наступний запис.

• **function SetEnd: integer**

- Встановити курсор на останній запис.

- **function GetPrev: integer**
 - Встановити курсор на попередній запис.
- **function GetKey: integer**
 - Встановити курсор на запис у відповідності зі значенням ключа **UniqueKeys**.
- **function Delete: integer**
 - Видаляє поточний запис в реєстрі.
- **procedure BeginUpdate procedure BeginUpdate**
 - Включає режим редагування реєстру, при якому дії з функціями роботи з реєстром не відображаються на екрані.
- **procedure EndUpdate**
 - Відключає режим редагування реєстру.
- **function ShowTable: integer**
 - Перевідає реєстр.

Функції роботи з полями:

- **function ParamField (FieldName: String, FieldType: String, FieldLen: Byte, FieldScale: Byte, Value: String): integer**
 - Заповнює поле-параметр XML - реєстра.
 - **FldName** - ім'я поля. Повинна відповідати імені в розділі <params>.
 - **FldType** - тип поля. Допустимі типи: "STRING", "INT", "NUM", "DATE", "TIME".
 - **FldLen** - довжина поля для типу STRING.
 - **FldScale** - місце точки для типу NUM.
 - **Value** - значення параметра.
- **function GetFieldValue (FieldName: String): String**
 - Отримати значення поля по імені
- **function SetFieldValue (FieldName: String, Value: String): integer**
 - Встановити значення поля по імені. Використовується для заповнення полів ключа. Поля записів в таблиці не змінює.
- **function SetFieldModifiable (FieldName: String, Modifiable: boolean): Boolean**
 - Встановлюється можливість редагування зазначеного поля в реєстрі. Працює тільки при **ReadOnly = false**.
- **functionfunction Set Sort (FieldNames: String): integer**
 - Встановити сортування по імені полів. **FieldNames** заповнюється наступним шаблоном: ім'я поля, напрямок сортування, ..., ім'я поля, напрямок сортування. Напрямок сортування може бути або по зростанню: 'A', або по спадаючій: 'D'
- **AllowSort (Allow: BOOLEAN): integer**
 - Дозволити / заборонити зміну сортування.

Гарячі клавіші:

- **F7** - Пошук;
- **F11** - Фільтр;
- **Alt + F11** - Фільтр за поточним;
- **F12** - Налаштування сортування.

Властивості:

- **ReadOnly** - Ознака реєстру «тільки для читання». Для редагування записів прямо в реєстрі встановити в false;
- **Insertable** - Ознака можливості вставки. При установці в true при русі курсору нижче останнього запису створюється нова;
- **CaptionLines** - властивість визначає кількість рівнів в шапці таблиці. Заповнюється шапка як і раніше в XML-файл у властивості name тега <field>. Приклад: name = "Загальна шапка | Поле ". Рівні шапки поділяються символом «| ». Однакову назву декількох колонок об'єднує їх в одну;
- **FixedCols** - Це властивість вказує кількість зафіксованих колонок з лівого краю таблиці.

Приклад:

// Робимо реєстр, що редагується

GridXML 1. Refresh;

GridXML1.ReadOnly: = false;

GridXML1.Insertable: = true;

GridXML1.SetFieldModifiable ('PtnCd', true); // Поле контрагента - редагується

procedure GridXML1OnModifyField (Sender: TISPROCComponent);

var

PtnCd, PtnNm: String;

begin

PtnCd: = GridXML1.SetFieldValue ('PtnCd'); // Отримали новий код контрагента

//.....Отримали по ньому найменування, наприклад окремим запитом

```
GridXML1.SetFieldValue (PtnNm); // Записали в реєстр найменування  
end;
```

1.1.2.3.1. код програми:

```
procedure Form1OnShow (Sender: TISPROC oponent);  
  
var  
  
ret: integer;  
  
begin  
  
// Запуск  
  
GridXML1.XMLFileDescription.MdID: = 555;  
  
GridXML1.XMLFileDescription.RstID: = 0;  
  
GridXML1.XMLFileDescription.QueryID: = 1;  
  
GridXML1.XMLBaseDescription.MarkAbsField: = 'TestRcd';  
  
GridXML1.XMLBaseDescription.Mrk: = 20000;  
  
GridXML1.XMLBaseDescription.UniqueKeys: = 'TestRcd';  
  
ret: = GridXML1.Init;  
  
if (ret> 0) then  
  
begin  
  
// Параметр 0  
  
ret: = GridXML1.ParamField ( 'TestID', 'INT', 0, 0, '0');  
  
if (ret> 0) then  
  
GridXML1.Refresh;  
  
end  
  
end;  
  
// Кнопка "наступний запис"
```

```

procedure But ton1OnClick (Sender: TISPROComponent);

begin

GridXML1.GetNext;

end;

// Кнопка "попереднійзапис "

procedure Button2OnClick (Sender: TISPROComponent);

begin

GridXML1.GetPrev;

end;

procedure GridXML1OnDbClick (Sender: TISPROComponent);

var

Rcd, ret: integer;

begin

// Отримуємо Ід поточного запису

Rcd := StrToInt (GridXML1.GetFieldValue ( 'TestRcd'));

// Щось робимо із записом

// .....

// ODBCQuery 1.Open;

// .....

// Оновлюємо запис з сервера

ret := GridXML1.ParamField ( 'TestID', 'INT', 0, 0, IntToStr (Rcd)); //Ід нужной записи // Ід потрібного
запису

if (ret> 0) then

Begin

```

```

GridXML1.BeginUpdate;

GridXML1.Delete; // Видаляємо

GridXML1.RefreshNoClear; // оновлюємо один запис

GridXML1.EndUpdate;

GridXML1.ParamField ( 'TestID', 'INT', 0, 0, '0'); // Обнуляєм // Обнуляємо параметр параметр

GridXML1.SetFieldValue ( 'TestRcd', IntToStr (Rcd)); // позиціонуємо на оновленому запису

GridXML1.GetKey;

End;

end;

procedure GridXML1OnKeyDown (Sender: TISPROComponent; var Key: Word; Shift: Integer);

begin

if (Key = 13) then // При натисканні Enter дані поточного запису:

ShowMessage ('Номер документа:' + GridXML 1. GetFieldValue ('TestNmr') + 'Сума:' + GridXML 1.
GetFieldValue ('TestSm'));

end; begin

end.

```

1.1.2.3.2. Використовуваний XML-реєстр:

U555_0_001.xml

```

<? xml v ersion = "1.0" encoding = "windows-1251"?>

< Query name = "Документи призначені для користувача">

<Result - fields>

<field name = "Rcd" as = "TestRcd" olap = "" progressbar = "" progressbarright = "" progressbarcolor = ""
olapicon = "" calculated = "true" scale = "0" type = "int" prgres = " "p rgtype = " 2 ">

<program> CashDoc_Rcd </ program> </ field>

```

```
<field name = "Номер" as = "TestNmr" olap = "" progressbar = "" progressbarright = "" progressbarcolor = "" olapicon = "" calculated = "true" scale = "20" type = "string" prgres = " " prgtype = " 2 ">
```

```
<program> CashDoc_DocNmr </ program> </ field>
```

```
<field name = "Дата" as = "TestDat" olap = "" progressbar = "" progressbarright = "" progressbarcolor = "" olapicon = "" calculated = "true" scale = "20" type = "string" prgres = " " prgtype = " 2 ">
```

```
<program> CashDocR.CashDoc_DocDat </ program> </ field>
```

```
<field name = "Контрагент" as = "TestPtn" olap = "" progressbar = "" progressbarright = "" progressbarcolor = "" olapicon = "" calculated = "true" scale = "20" type = "string" prgres = " " prgtype = " 2 ">
```

```
<program> CashDoc_Ptn </ program> </ field>
```

```
<field name = " Сума " as = "TestSm" olap = "" progressbar = "" progressbarright = "" progressbarcolor = "" olapicon = "" calculated = "true" scale = "20" type = "string" prgres = " " prgtype = " 2 ">
```

```
<Program> Round (CashDoc_Sm, Svl_Acc) </ program> </ field>
```

```
</ Result-fields>
```

```
<Tables>
```

```
FROM CashDocR
```

```
LEFT JOIN Svl on SVL_Rcd = CashDoc_Val
```

```
</ Tables>
```

```
<Where>
```

```
(CashDoc_Rcd =: TestID OR: TestID = 0)
```

```
</ Where>
```

```
<Params>
```

```
<param name = "ID документа " id = "TestID" type = "int" />
```

```
</ Params>
```

```
</ Query>
```

1.1.2.4. Компонент TISPROVirtualTree

Компонент **TISPROVirtualTree** призначений для роботи з деревом з колонками. У дизайнера вікон на вкладці **Розширені**.

Функції:

Функції створення структури дерева:

- **procedure AddNode (ID: Int64; ParentID: Int64; Text: String)**
 - Додає гілку з ім'ям **Text** в дерево, ID якій повинно бути унікальним.
 - **ParentID** - ID гілки, в яку буде додаватися нова гілка.
- **procedure AddColumn (Name: String)**
 - Додає в дерево нову колонку з ім'ям **Name**. Текст в гілках нової колонки встановлюється з першої колонки відповідної гілки.
- **procedure AddNumericColumn (Name: String; Triads: BOOLEAN)**
 - Додає колонку числового типу. **Triads** - встановлює відображення числа в вигляді триад.
- **procedure AddDateTimeColumn (Name: String; MaskType: Integer)**
 - Додає колонку типу «Дата». **MaskType** - встановлює тип відображення дати (1 - з часом, 2 - без часу)

Функції заповнення дерева:

- **procedure SetText (NodeID: Int64; ColumnIndex: Int64; Text: String)**
 - Встановлює текст **Text** в осередку гілки з номером **NodeID** і колонки з номером **ColumnIndex** (нумерація починається з 0).
- **procedure SetNumericValue (NodeID : Int 64; ColumnIndex : Int 64; Value : Double)** Аналогічно з **SetText** встановлює значення типу Double для колонки з відповідним типом.
- **procedure SetDateTimeValue (ID : Int 64; ColumnIndex : Int 64; Value: TDateTime)** Аналогічно з **SetText** встановлює значення типу TDateTime для колонки з відповідним типом.
- **procedure SetColumnName (ColumnIndex: Integer; Name: String)**
 - Встановлює ім'я **Name** колонки з номером **ColumnIndex**.
- **procedure SetNodeCheck (ID: Int 64; Checked: Boolean)** Встановлює або знімає позначку з гілки з номером ID.

Функції навігації по дереву:

- **procedure SetFocusedNode (ID: Int64)**
 - Встановлює курсор на гілці з номером ID, при цьому фокус колонки не змінюється.

- **procedure SetFocusedColumn (ColumnIndex: Integer)**
 - Встановлює курсор на колонці з номером ColumnIndex, при цьому фокус гілки не змінюється.
- **procedure ExpandNode (ID: Int64)**
 - Розкриває гілку з номером ID.
- **procedure CollapsNode (ID: Int64)**
 - Згортає гілку з номером ID.
- **procedure ExpandAll ()**
 - Розкриває всі гілки.
- **procedure CollapsAll ()**
 - Згортає всі гілки.

Функції зворотного зв'язку з деревом

- **function GetText (NodeID : Int 64; ColumnIndex : Int 64): String** Повертає текст в осередку, яка розміщена в гілці з номером NodeID і колонці з номером ColumnIndex.
- **'function GetFocusedText (): String'**
 - Повертає текст у виділеній комірці.
- **function GetSelectedNodeID (): Int64**
 - Повертає номер гілки, на якій знаходиться курсор.
- **function GetSelectedColumnIndex (): Int64**
 - Повертає номер колонки, на якій знаходиться курсор.
- **function GetNodeCheck (ID: Int64): Boolean**
 - Повертає true або false , що відображають відзначена чи гілка дерева.
- **function GetNodeParentID (ID : Int 64): Int 64** Повертає ID батька гілки з номером ID.
- **function GetNodeHeight (ID : Int 64): Integer** Повертає висоту Ноди з номером ID.
- **Функція function GetCurrentField: String**
 - Повертає ім'я поля поточної колонки.

Приклад:

```
procedure GridXML1OnDbClick (Sender: TISPROComponent);
begin
  ShowMessage (GridXML1.GetCurrentField + " + GridXML1.GetFieldValue (GridXML1.GetCurrentField));
end;
```

Функції:

- **function SetColumnWidth (Індекс колонки, Ширина колонки)**
 - Приклад використання: VirtualTree1.SetColumnWidth (0,200);

Функції циклічного перебору структури дерева:

- **function GetFirstNodeID (): Int64**
 - Повертає ID першої гілки.
- **function GetNextNodeID (ID : Int 64): Int 64**
 - Повертає ID наступної за переданої в параметрі функції гілки.
- **function GetNodeCount (): Int64**
 - Повертає кількість гілок в дереві.
- **function GetColumnCount (): Int64**
 - Повертає кількість колонок в дереві.

Функції Редагування дерева:

- **AllowManualEditing** - дозволяє режим редагування
- **procedure EditFocusedNode**
 - Включає режим редагування поточної комірки.
- **procedure SetColumnWidth (ColumnIndex : Integer ; Width : Integer)**
 - Установка ширини обраної колонки.
- **procedure SetColumnAlignment (ColumnIndex: Int64; CellAlignment: string)**
 - Установка вирівнювання в осередку. Варіанти: 'caRight', 'ca Center', 'ca Left'
- **procedure SetColumnPrecision (ColumnIndex: Int64; Precision: integer)**
 - Установка точності чисел (кількості знаків після коми). Зауваження : кількість знаків після коми обмежує число тільки при відображенні, введене число зберігається в пам'яті.
- **procedure SetColumnStyle (ColumnIndex : Int 64; BackgroundColor, TextColor : TColor ; TextSize : Integer ; Bold, Italic : Boolean)**
 - Установка стилю колонки: колір фону, тексту, розмір тексту (в пікселях), стиль шрифту (Bold - жирний, Italic - похилий)
- **procedure SetCellStyle (ID: Int64; ColumnIndex: Int64; BackgroundColor, TextColor: TColor; TextSize: Integer; Bold, Italic: Boolean)**
 - Аналогічно **SetColumnStyle**, але застосовувана к до осередку. Стиль комірки має перевагу перед стилем колонки.
- **procedure SetColumnMask (ColumnIndex: Int64; MaskType: Integer)**
 - Установка типу відображення дати (застосовна к до колонці типу **Дата**). **MaskType**: 1 - дата і час, 2 - тільки дата

- **procedure DeleteNode (ID : Int 64)**
 - Видалення гілки дерева з ідентифікатором ID Якщо видаляється батьківський вузол то всі дочірні вузли теж видаляються.
- **procedure Clear**
 - Очищення дерева
- **procedure SetNodeHeight (ID: Int64; value: Integer)**
 - Установка висоти Ноди.

Властивості:

SummaryHeight - висота підсумкового поля

Можливості:

- швидкий пошук по колонці, на якій стоїть фокус;
- форматованого виведення чисел і дат, а також налаштування форматування комірки;
- форматування комірки.

Приклади:

• **Код програми:**

• **заповнення дерева**

```
procedure Button1OnClick (Sender: TISPROComponent);
```

```
var
```

```
i: Int64;
```

```
begin
```

```
VirtualTree1.AddNode (1,0, '1');
```

```
VirtualTree1.AddNode (2,1, '11');
```

```
VirtualTree1.AddNode (11,0, '2');
```

```
VirtualTree1.AddNode (12,2, '111');
```

```
VirtualTree1.AddNode (17,2, '112');
```

```
VirtualTree1.AddNode (18,11, '21');
```

```
VirtualTree1.AddColumn ('col');
```

```
end;
```

- **встановити тест в певній клітинці**

```
procedure Button2OnClick (Sender: TISPROComponent);
```

```
begin
```

```
VirtualTree1.SetText (2,1, 'dcdvdv');
```

```
end;
```

- **встановити фокус в вибраній комірці**

```
procedure Button5OnClick (Sender: TISPROComponent);
```

```
begin
```

```
VirtualTree1.SetFocusedColumn (1);
```

```
VirtualTree1.SetFocusedNode (11);
```

```
end;
```

- **отримати текст комірки**

```
procedure Button6OnClick (Sender: TISPROComponent);
```

```
begin;
```

```
ShowMessage (VirtualTree1.GetText (2, 1));
```

```
end;
```

Для компонента **TISPROVirtualTree** реалізовані події, які реагують на прокрутку колеса миші:

- **OnMouseWheel;**
- **OnMouseWheelUp;**
- **OnMouseWheelDown.**

Параметри відповідають аналогічним подіям в **VCL**, крім **MousePos: TPoint**, який перетворений в **MousePos.X, MousePos.Y**. Спрацьовують, тільки якщо **VirtualTree** в фокусі, але знаходження курсору на ньому не обов'язково.

1.1.2.5. Компонент TISPROXML

Модуль **Конструктор** - вкладка **Ресурси**. Подвійне натискання на компоненті відкриває дизайнер. Створений в дизайнері XML-документ зберігається всередині компонента.

XML--функції:

Відобразити дизайнер: procedure ShowDesigner

Завантажити з файлу: procedure LoadFromFile (const FileName: string)

Зберегти в файл: procedure SaveToFile (const FileName: string)

Завантажити з рядка: procedure ReadFromString (const Text: String)

Зберегти в рядок: function WriteToString: String

Створити новий xml: procedure New

Додати вузол до зазначеного: function NodeInsertBefore (FocusedNode: TISsXMLNode): TISsXMLNode

Додати вузол після зазначеного: function NodeInsertAfter (FocusedNode: TISsXMLNode): TISsXMLNode

Додати підлеглий: function NodeInsertChild (FocusedNode: TISsXMLNode): TISsXMLNode

Властивості:

Кореневий вузол: Root

Вузол TISsXMLNode

Властивості:

НаименованиеНайменування: Name: String

ЗначениеЗначення: Value: String

Кількість підлеглих вузлів: NodeCount: Integer;

Отримати вузол за індексом: Nodes [i]: TISsXMLNode (тільки для читання)

Кількість атрибутів: AttributeCount: Integer

Отримати атрибут за індексом: Attributes [i]: TISsXMLAttribute (тільки для читання)

Значення атрибута по імені: AttributeValueByName [Name]: String

Функції:

Додати атрибут: procedure AttributeAdd (Name: String, Value: String)

Видалити атрибут за індексом: procedure AttributeDelete (Index: integer)

Додати вузол: procedure NodeAdd (Name: String, Value: String)

Видалити вузол за індексом: procedure NodeDelete (Index: integer)

Знайти вузол на ім'я: function FindNode (const NodeName: String): TISXMLNode

Атрибут TISXMLAttribute

Наименование, Найменування: Name: String

ЗначениеЗЩ, Значення: Value: String

Приклад використання:

```
procedure Button1OnClick (Sender: TISPROComponent);
```

```
var
```

```
Node: TISXMLNode;
```

```
i: integer;
```

```
begin
```

```
XML1.Root.AttributeAdd ( 'A0', '1');
```

```
Node := XML1.Root.NodeAdd ( 'Node1', '2');
```

```
Node.AttributeAdd ( 'A1', '2');
```

```
XML1.ShowDesigner;
```

```
for i := 0 to XML1.Root.NodeCount-1 do
```

```
ShowMessage (XML1.Root.Nodes [i] .Name + " + XML1.Root.Nodes [i] .Value);
```

```
end;
```

1.1.2.6. Компонент TISPROPopupMenu

Компонент створення контекстного меню. Знаходиться на вкладці **Стандартні**.

Створення структури меню. Створення пунктів і підпунктів меню проводиться натисненням правої кнопки миші по значку контекстного меню або значку елемента меню у вікні структури модуля і вибору відповідного пункту.

- Натискання на значок контекстного меню. При виборі пункту «**Підпункт**» створиться підпункт меню в кінці першого рівня контекстного меню.
- Натискання на значок пункту меню.

1. **Пункт**: створиться пункт меню перед обраним пунктом на цьому ж рівні;
2. **Підпункт** : створиться пункт меню, як підпункт обраного в кінці списку.

Пункти **Вгору** і **Вниз** дозволяють переміщати пункти меню відповідно вгору або вниз.

Властивості контекстного меню:

- **ImageList**: підключення **ImageList**, створеного в **Ресурсах** , до меню;
- Для підключення контекстного меню до компонентів, в їх властивості **PopupMenu** потрібно вибрати необхідне меню.

Властивості пунктів меню:

- **AutoCheck**: якщо властивість встановлено в true, то при натисканні на пункті меню в запустив додаток, властивість змінюється на протилежне;
- **Bitmap**: біля пункту меню малюється вибране зображення;
- **Checked**: якщо властивість встановлено в true, то біля пункту меню малюється значок галочки;
- **ImageIndex**: якщо до контекстного меню підключений **ImageList**, то біля пункту меню малюється зображення під номером ImageIndex в підключеному **ImageList**-ті.

1.1.2.7. Компонент TISPRMainMenu

Компонент створення **Головне меню**. Знаходиться на вкладці **Стандартні**.

Створення структури меню. Створення пунктів і підпунктів меню проводиться натисненням правої кнопки миші по значку головного меню або значку елемента меню у вікні структури модуля і вибору відповідного пункту.

- Натискання на значок головного меню:
 - При виборі єдиного можливого пункту **Підпункт** створиться підпункт меню в кінці першого рівня головного меню.
- Натискання на значок пункту меню:
 1. **Пункт**: створиться пункт меню перед обраним пунктом на цьому ж рівні;
 2. **Підпункт**: створиться пункт меню, як підпункт обраного в кінці списку.

Властивості головного меню:

- **ImageList**: підключення **ImageList**, створеного в **Ресурсах**, до головного меню.

Властивості пунктів меню:

- **AutoCheck**: якщо властивість встановлено в true, то при натисканні на пункті меню в запустив додаток, властивість змінюється на протилежне;
- **Bitmap**: біля пункту меню малюється вибране зображення;
- **Checked**: якщо властивість встановлено в true, то біля пункту меню малюється значок галочки;
- **ImageIndex**: якщо до головного меню підключений **ImageList**, то біля пункту меню малюється зображення під номером **ImageIndex** в підключеному **ImageList**-ті.

1.1.2.8. Компонент TISPROBARCODE (штрих-код)

Знаходиться на вкладці **Розширені**. Компонент TISPROBarcode.

Властивості:

- **BarType (TISxBarcodeType)** - тип кодування штрих-коду:
 1. bcCode _2_5_ interleaved;
 2. bcCode _2_5_ industrial;
 3. bcCode _2_5_ matrix;
 4. bcCode 39;
 5. bcCode 39 Extended;
 6. bcCode 128;
 7. bcCode 128 A;
 8. bcCode 128 B;
 9. bcCode 128 C;
 10. bcCode 93;
 11. bcCode 93 Extended;

12. bcCodeMSI;
13. bcCodePostNet;
14. bcCodeCodabar;
15. bcCodeEAN 8;
16. bcCodeEAN 13;
17. bcCodeUPC _ A;
18. bcCodeUPC _ E 0;
19. bcCodeUPC _ E 1;
20. bcCodeUPC _ Supp 2;
21. bcCodeUPC _ Supp 5;
22. bcCodeEAN 128;
23. bcCodeEAN 128 A;
24. bcCodeEAN 128 B;
25. bcCodeEAN 128 C.

- **CalcChecksum (Boolean)** - обчислювати / НЕ обчислювати контрольну суму;
- **Rotation (Integer)** - поворот штрих-коду на вказаний кут, кратний 90 °;
- **ShowText (Boolean)** - показувати / НЕ показувати кодований текст під штрих-кодом;
- **Text (String)** - кодований текст;
- **WideBarRatio (Extended)** - співвідношення ширини товстого і тонкого штриха;
- **Zoom (Extended)** - масштаб.

1.1.2.9. Компонент TISPRO Q RCODE (штрих-код)

Знаходиться на вкладці **Розширені**. Компонент TISPROQRCode.

Властивості:

- **Encoding(TQRCodeEncoding)** - тип кодування QR -коду;
 1. qrAuto;
 2. qrNumeric;
 3. qrAlphanumeric;
 4. qrISO88591;
 5. qrUTF8NoBOM;
 6. qrUTF8BOM;
 7. qrShift_JIS.
- **FontScaled (Boolean)** - масштабувати / НЕ масштабувати підпис QR - коду;
- **Rotation (Integer)** - поворот штрих-коду на вказаний кут, кратний 90 °;
- **ShowText (Boolean)** - показувати / НЕ показувати кодований текст під штрих-кодом;

- **Text (String)** - кодований текст;
- **PixelSize (Integer)** - розмір пікселю;
- **Zoom (Extended)** - масштаб.

1.1.2.10. Компонент TISPROMAIL

Знаходиться на вкладці **Розширені**. Компонент TISPROMail.

Функції:

- **function Connect (TargetHost: AnsiString; TargetPort: AnsiString; UserName: AnsiString; Password: AnsiString): boolean** - підключення к до хосту поштової служби (повертає true при вдалому підключенні);
- **procedure Disconnect** - відключення от від хоста;
- **function GetEMailsCount : Integer** - повертає кількість листів в поштовій скриньці;
- **function GetEMail (EMailIndex : Integer ; Delete : boolean): Boolean** - повертає true у разі успішного завантаження листи. У функцію передається:
 1. номер листа по списком;
 2. Чи видаляти лист з сервера після зачитуванні.
- **procedure SaveAttachByPath (i : Integer ; path : String)** - отримати додаток до листа (в разі наявності в тексті листа html розмітки у вкладення додається html -файл з цим вмістом). Параметр **path** визначає шлях, по якому зберігати файл (якщо шлях не зазначений або такого шляху не існує, то з'явиться вікно вибору шляху);
- **function GetAttachesCount : Integer** - повертає кількість вкладень в листі.

Властивості:

- **EMailTo** - ім'я одержувача листа;
- **EMailFrom** - ім'я відправника листа;
- **EMailSubject** - тема листа;
- **EmailText** - текст листа.

Код програми отримання імені одержувача, імені відправника, теми і тексту листа:

```
procedure Button1OnClick (Sender: TISPROComponent);
```

```
var
```

```
i, j: integer;
```

```
begin
```

```

if (Mail1.Connect ('mail.intelserv.kiev.ua', '110', '*****', '*****')) then

begin

for i: = 0 to Mail1.GetEMailsCount - 1 do

begin

if (Mail1.GetEmail (i, false)) then

begin

ShowMessage ('To:' + Mail1.EMailTo);

ShowMessage ('From:' + Mail1.EMailFrom);

ShowMessage ('Subject:' + Mail1.EMailSubject);

ShowMessage ('Text:' + Mail1.EMailText);

for j: = 0 to Mail1.GetAttachesCount - 1 do

Mail1.SaveAttachByPath (j, 'D: \ tmpfiles ');

end;

end;

end;

end;

```

1.1.3. Користувальницькі реєстри

1.1.3.1. Опис структури XML-файлу

Для створення призначеного для користувача XML - реєстру в першу чергу необхідно створити XML - файл з описом полів, таблиць та умов відображення даних. Приклад призначеного для користувача реєстру:

```

<? Xml version = "1.0" encoding = "windows-1251"?>

<Query name = "Документи призначені для користувача">

<Result-fields>

```

```
<Field name = "Rcd" as = "CashDoc_Rcd" olap = "" progressbar = "" progressbarright = ""
progressbarcolor = "" olapicon = "" calculated = "true" scale = "0" type = "int" prgres = " " prgtype = " 2
"visible = " false ">
```

```
<Program> CashDoc_Rcd </ program> </ field>
```

```
<Field name = "Homep" as = "CashTestNmr" olap = "" progressbar = "" progressbarright = ""
progressbarcolor = "" olapicon = "" calculated = "true" scale = "20" type = "string" prgres = " " prgtype = " 2
">
```

```
<Program> CashDoc_DocNmr </ program> </ field>
```

```
<Field name = "Датум" as = "CashTestDat" olap = "" progressbar = "" progressbarright = ""
progressbarcolor = "" olapicon = "" calculated = "true" scale = "20" type = "string" prgres = " " prgtype = " 2
">
```

```
<Program> CashDocR.CashDoc_DocDat </ program> </ field>
```

```
<Field name = "Контрагент" as = "CashTestPtn" olap = "" progressbar = "" progressbarright = ""
progressbarcolor = "" olapicon = "" calculated = "true" scale = "20" type = "string" prgres = " " prgtype = " 2
">
```

```
<Program> CashDoc_Ptn </ program> </ field>
```

```
<Field name = "Сума" as = "CashTestSm" olap = "" progressbar = "" progressbarright = ""
progressbarcolor = "" olapicon = "" calculated = "true" scale = "20" type = "string" prgres = " " prgtype = " 2
">
```

```
<Program> Round (CashDoc_Sm, Svl_Acc) </ program> </ field>
```

```
</ Result-fields>
```

```
<Tables>
```

```
FROM CashDocR
```

```
LEFT JOIN Svl on Svl_Rcd = CashDoc_Val
```

```
LEFT JOIN SprPls ON SprPls_Rcd = CashDoc_Sch
```

LEFT JOIN Jr ON Jr_Rcd = CashDoc_JrnRcd

LEFT JOIN DogSprOtv ON DogSprOtv_Rcd = CashDoc_Otv

LEFT JOIN SOu ON Ou_Rcd = CashDoc_CdOu

</ Tables>

<Where>

(CashDoc_Rcd =: prmValidRec OR: prmValidRec = 0)

</ Where>

<Params>

<Param name = "ID документа" id = "prmValidRec" type = "int" />

</ Params>

</ Query>

Опис параметрів файлу:

- *Query name* - назва реєстру. Відобразатиметься в списку вікна **Види реєстру** (комбінація клавіш **Alt + O**) ;
- Блок *< result - fields > </ result - fields >* - між цими тегами йде опис всіх полів, які є в реєстрі;
- Блок *<field name = "Номер" as = "CashTestNmr" olap = "" progressbar = "" progressbarright = "" progressbarcolor = "" olapicon = "" calculated = "true" scale = "20" type = "string" prgres = "" prgtype = "2"> <program> CashDoc_DocNmr </ program> </ field>* - опис поля в реєстрі:
 - *name = "Номер"* - назва поля, буде відображатися в шапці таблиці;
 - *as = " CashTestNmr "* - назва поля, з яким надалі можна буде працювати з калькуляцій;
 - *scale = "20"* - довжина поля, для поля типу int довжина завжди дорівнює нулю;
 - *type = " string "* - тип поля . Може бути string, int;
 - *prgtype = "2"* - тип обробки програми . 1 - BasicScript, 2 - SQL запит;
 - *visible = " false "* - вказує чи буде поле видимим в реєстрі для користувача;
 - *<program> CashDoc_DocNmr </ program>* - програмаполучення отримання поля;
- Блок *< tables > </ tables >* - описуються таблиці, з яких потрібно отримувати поля, а так же ключі зв'язування таблиць. Синтаксис аналогічний SQL запитам;

- Блок `< where > </ where >` - в цьому блоці описуються умови вибірки даних із зазначених в тезі `< tables >` таблиць. У цьому блоці повинна бути обов'язково вказана рядок (`CashDoc_Rcd =: prmValldRec OR: prmValldRec = 0`), де `CashDoc_Rcd` це унікальний запис (`Rcd`) в таблиці ;
- Блок `< params > </ params >` містить список переданих параметрів з **Комплексу** в запит. У всіх призначених для користувача реєстрах блок повинен містити як мінімум один параметр, як в прикладі;

Імена для користувача XML файлів завжди повинні починатися з букви U (наприклад, U047_0_001.xml). Створений файл потрібно помістити в папку XML серверної частини **Комплексу**.

1.1.3.2. Додавання користувальницького реєстру

Для створення нового користувальницького реєстру необхідно в модулі **Конструктор** на вкладці **Реєстри** натиснути пункт меню **Створити** (клавіша **Ins**). У діалоговому вікні в полі **Номер** вказати унікальний код реєстру, в полі **НаименованиеНайменування** - назва призначеного для користувача реєстру. В поле **Файл** вибрати файл XML - реєстру, назва якого починається на символ U і знаходиться в папці XML серверної частини **Комплексу**. В поле **Таблиця** потрібно вказати головну таблицю, з якої буде проводитися побудова реєстру, в полі **Ід таблиці** вказати поле, яке містить унікальний запис головної таблиці (`Rcd` таблиці). В поле **Маркер** ввести код для маркування списку в реєстрі. Вводяться числа від 10000, значення повинно бути унікальним для всіх призначених для користувача реєстрів. В поле **Форма** вводиться код екранної форми , значення поля може бути від 5000 до 10000. Це поле служить для зазначення груп и звітів, доступних користувачеві для друку. При натисканні на кнопки в розділі **Програми** можна створювати для користувача логіку обробки таких дій як створення, редагування, копіювання, видалення записів з реєстру. Програма групової операції буде відпрацьована для кожного зазначеного запису в реєстрі. Обробка призначеної для користувача події запускає написану програму для обраного запису реєстру після натискання комбінації клавіш **Alt + U**.

1.1.3.3. Редактор XML-файлу

Для створення XML-реєстрів можна використовувати редактор XML-файлів. Редактор відкривається з вікна створення / редагування користувальницького реєстру (вкладка **Реєстри** при виборі пунктів **Створити / Відкрити**) по кнопці в полі **Файл**. Редактор відображає ієрархічну структуру XML-файлу з візуальним виділенням різних елементів. Може створювати, змінювати, видаляти елементи, атрибути, коментарі. Відображати текст XML-файлу з підсвічуванням синтаксису. Має можливість редагувати текст XML-файлу. Відображає допоміжну (відладочну) інформацію: підказки, помилки і попередження. Редактор може створювати новий XML файл зі структурою реєстрів **Комплексу**. Є вікна редагування елементів реєстрів **Комплексу**.

1.1.3.4. Користувальницькі таблиці

Для створення нової користувальницької таблиці необхідно в меню модуля **Конструктор** на вкладці **Таблиці** вибрати пункт меню **Створити** (клавіша **Ins**). У вікні потрібно вказати назву таблиці, яка буде створена в базі даних (всі призначені для користувача таблиці мають префікс U _), найменування таблиці, яке буде відображатися в конструкторі. Так само потрібно вказати тип таблиці: монопольна або спільна. Якщо обраний тип **монопольна**, то таблиця створюється в базі підприємства і доступ до неї є тільки з цього підприємства. Якщо обраний тип **спільна**, то таблиця створюється в системній базі і може бути доступною з усіх підприємств

У вікні створення таблиці є дві вкладки. На першій вкладці можна створювати поля, в другій - індекси. При створенні нового поля в таблиці потрібно вказати ідентифікатор (з цим ім'ям буде створено поле в таблиці). **Найменування** - опис поля, **default** - значення поля за замовчуванням (якщо при вставці нового запису в таблицю поле буде порожнім, то воно автоматично буде заповнено значенням за замовчуванням), клас - тип поля, аналогічні з типами в полях, довідниках користувача. Довжина поля - кількість символів в строковому полі. Обмеження - можливі обмеження, які накладаються на це поле. Параметр **Identity** вказує на те, що сервер баз даних буде виділяти для поля унікальний номер. Параметр **Nullable** вказує, чи може бути це поле порожнім (null).

Після створення полів таблиці можна перейти до створення індексів. На вкладці **Індекси** використовувати кнопку **Створити** (клавіша **Ins**). У вікні в лівому списку буде вибір полів таблиці. Необхідно встановити курсор на поле, по якому повинен будуватися індекс і натиснути кнопку **F5**. Поле з'явиться в правому списку, по цьому полю буде будуватися індекс. Індекс можна побудувати як по одному, так і по декількох полях таблиці. В правому списку можна вибрати за яким принципом буде будуватися індекс (наростання або спадання) встановленням параметра **По убуванню** біля імені індексу. Після вибору поля потрібно вказати тип індексу: **Primary, Dupl, NoDupl** (основний, дубльований, що не дубльований).

Перед виходом з вікна редагування таблиця автоматично буде створена і конвертована.

Щоб додати новий користувальницький модуль в головне меню **Комплексу** необхідно зайти у вікно редагування меню (модуль **Користувачі і ролі**). У лівому списку встановити курсор на пункті **Зовнішні завдання** і натиснути кнопку **Створити** (клавіша **Ins**). У діалоговому вікні вказати найменування призначеного для користувача модуля, після чого натиснути кнопку **Текст**. У вікні написати одну з описаних нижче функцій:

- **RUNUSERRST (uint2)** - запустити реєстр користувача, вхідний параметр це код реєстру;
- **RU NUSERMODULE (uint4)** - запустити модуль користувача, вхідний параметр це код модуля.

Використання цих функцій не обмежується модулем **Користувачі і ролі**, вони можуть бути використані в будь-якій калькуляції або програмі користувача в **Комплексі**.

1.1.3.5. Експорт та імпорт призначених для користувача вікон, таблиць та реєстрів

При необхідності на кожній з вкладок можна зробити експорт відповідних вікон, реєстрів і таблиць і імпорт.

Щоб зробити експорт вікон, таблиць або реєстрів необхідно вибрати потрібні записи в списку і в меню **Реєстр** вибрати пункт **Експорт**. У діалоговому вікні потрібно вибрати папку, в яку будуть вивантажені записи. Під час вивантаження таблиць крім опису так само вивантажуються і вміст таблиці. Дані знаходяться в окремому файлі, який називається, так само як і таблиця.

Щоб імпортувати вікна, таблиці або реєстри потрібно вибрати потрібні записи в списку і в меню **Реєстр** вибрати пункт **Імпорт**. Ці записи імпортуються в одному з режимів **Додати** або **Замінити**. У режимі **Додати** імпортовані записи додаються до існуючого списку і присвоюються нові номери, а в режимі **Замінити** проводиться заміна записів з однаковими номерами. У діалоговому вікні потрібно вибрати папку, в якій знаходяться файли для імпорту. Якщо в папці XML серверної частини **Комплексу** вже знаходиться XML - файл з найменуванням, як у файлі для імпорту то імпорт проведений не буде. Це актуально і для назв таблиць.

1.1.3.6. Функції для роботи з користувальницькими вікнами і реєстрами

Список функцій **ISpro Комплексу** для роботи з користувальницькими вікнами і реєстрами:

- *RUNUSERRST (uint2)* - запустити реєстр користувача, вхідний параметр це код реєстру;
- *RUNUSERMODULE (uint4)* - запустити модуль користувача, вхідний параметр це код модуля;
- *REFRESHUSERRST ()* - оновити всі записи в призначеному для користувача реєстрі;
- *REFRESH1REC ()* - оновити поточний запис в призначеному для користувача реєстрі;
- *int GETFIRSTMARKID ()* - повертає ID першого зазначеного запису в реєстрі. Якщо жодний запис не вибрано, то функція поверне 0;
- *int GETNEXTMARKID ()* - повертає ID наступного зазначеного запису в реєстрі. Якщо проведений повний прохід по зазначеним записам, то функція поверне 0.

Додано створення і використання користувальницьких глобальних змінних:

- *SETVALUEFORNAME (const pchar, pStayFD fd)* - створити змінну з ім'ям *pchar*, в якій буде зберігатися значення *fd*, може бути переданий будь-який тип даних;
- *GETVALUEFORNAME (const pchar)* - отримати значення з змінної з ім'ям *pchar*, яка була оголошена раніше;
- *DELETEVALUEFORNAME (const pchar)* - видалити змінну з ім'ям *pchar* з пам'яті.

Функції зняття позначок з реєстру:

- *void CLAEALLMARK ()* - зняти всі позначки, які встановлені в реєстрі;
- *uint4 MARKUNMARKID (uint4 Id)* - відзначити / зняти позначку з запису реєстру.

Функція виклику калькулятора **Комплексу**:

- function **ShowCalculator (const AValue : Double)**
- *Double* – про **Ctrl + Enter** повертає результат обчислень, в іншому випадку - вхідний параметр.

Константи атрибутів файлів. Реалізовані функції:

- **FileSearch** - Здійснює пошук файлу в одній або більше папках;
- **FindInDir** - Повертає список файлів в каталозі по шляху, включаючи маску файлів, із зазначенням атрибутів файлів;
- **FileGetAttr** - Повертає атрибути файлу;
- **FileSetAttr** - Встановлює атрибути файлу.

Приклад запуску з передачею і отримання м параметрів:

- CALL SETVALUEFORNAME ("Name", "123")
- CALL SETVALUEFORNAME ("Name1", "OK")
- RUNUSERMODULE (3)
- MESSAGE (GETVALUEFORNAME ("Name"))

Результатом виконання цього прикладу буде оповіщення 123 після виходу з користувацького модуля з кодом 3.

1.1.3.7. Резервне копіювання

В резервну копію **Комплексу** потрапляють призначені для користувача вікна, реєстри і таблиці. Для того, щоб XML-файли користувацьких реєстрів потрапили в копію, вони повинні знаходитися в каталозі XML серверної частини **Комплексу**. В параметрах створення резервної

копії потрібно включити параметр **Зберігати звіти і види реєстри користувача**. В копію додаються описи користувацьких таблиць з каталогу PIC \ USR серверної частини і дані цих таблиць.

1.2. Конструктор (у web-інтерфейсі)

Модуль **Конструктор** у web-інтерфейсі **Комплексу ISpro** дозволяє налаштувати інтерфейс абсолютно всіх модулів **Комплексу** під себе.

Конструктор ISpro – зручний, ефективний і гнучкий інструмент адаптації до потреб конкретної компанії, конкретного користувача. У **Конструкторі** доступні можливості: змінювати розміри і розташування елементів будь-яких модулів системи, додавати свої вкладки, реєстри, графіки та інші необхідні компоненти. Склад, розміри, розташування – все це можна легко налаштувати візуально і відразу ж побачити результати своєї творчості.

При відкритті модуля **Конструктор** відображається реєстр створених шаблонів. На панелі інструментів розташовані кнопки:

- **Створити** ( або по пункту меню **Реєстр / Створити**) – створення нового шаблону;
- **Змінити** ( або по пункту меню **Реєстр / Змінити**) – редагування існуючого шаблону;
- **Видалити** ( або по пункту меню **Реєстр / Видалити**) – видалення шаблону;
- **Копіювати** ( або по пункту меню **Реєстр / Копіювати**) – створення копії шаблону;
- **Вихід** ( або по пункту меню **Реєстр / Вихід**) – вихід з модуля **Конструктор**.

Також по пункту меню **Реєстр** доступні команди:

- **Експорт модуля** – виконується експорт обраного модуля з реєстру. Файл експортованого модуля має розширення ***.isprowebmd**;
- **Імпорт модуля** – виконується імпорт модулів в **Комплекс**, для цього необхідно обрати місцезрештування файлу модуля з розширенням ***.isprowebmd**.

Реєстр ? Історія

В системі

Ад

adm

Код	↑	Найменування	↑	Тип	↑
21		Реєстр документів руху		Користувачський	
20		1111		Користувачський	
18		Дашбордин		Користувачський	
17		Тест 2		Користувачський	
16		Тест		Користувачський	
15		Реєстр документів руху		Користувачський	
14		Графики + реєстр копія		Користувачський	
12		111		Користувачський	
10		Графики + реєстр		Користувачський	
					Кількість записів 1

ТОВ "Підприємство"

ISpro: 8.0.001 | 001

Модуль Конструктор у web-інтерфейсі

По кнопці **Створити** або по пункту меню **Реєстр / Створити** – створюється шаблон модуля і у вікні **Створення шаблону** задається **Найменування шаблону** і натискається кнопка **ОК**.



Створення шаблону






Найменування шаблону

Модуль 1

ОК

ВІДМІНА

Створення шаблону модуля

При обранні модуля з реєстру шаблонів відкривається **Конструктор модулів**. Вікно **Конструктора модулів** містить зверху кнопки:

-  приховати / відобразити панель інструментів;
-  зберегти шаблон;
-  відміна дії;
-  попередній перегляд шаблону;
-  налаштування шаблону;
-  вихід з модуля **Конструктор**.

По кнопці **Налаштування шаблону** виконується налаштування шаблону модуля. На вкладці **Основні налаштування** вказується:

- **Найменування**;
- **Тип форми** – обирається значення зі списку **Документ** або **Діалог**;
- **Код документу**, обов'язкове поле для заповнення;
- **Код бізнес процесу**, обов'язкове поле для заповнення;
- **Код екранної форми**, обов'язкове поле для заповнення;
- **Зовнішній вид вкладок** – обирається значення зі списку **Звичайний** або **Навігація**.


Налаштування модуля




Основні налаштування

Події


 Найменування

Тип форми

Документ

▼


 Код документа


 Код бізнес процесу

ОК

ВІДМІНА

Налаштування модуля. Вкладка Основні налаштування

На вкладці **Події** по кнопці **Додати подію**  **Додати подію** з'являється вкладка **Отримати значення**, при натисненні якої розкривається список:

- **Дія** – обирається значення зі списку:
 - Отримати значення;
 - Змінити;
 - Відкрити модальне вікно;
 - Перейти на наступний рівень;
 - Змінити уявлення;
 - Розкрити гілку;
 - Отримати всю гілку до дочірнього елемента;
 - Запустити меню;
 - Створити запис;

- Оновити запис;
- Видалити запис;
- Опис колонок деревовидної структури;
- Відкрити на редагування;
- Відкрити на створення;
- Відкрити список.
- **ID** – обирається значення зі списку:
 - _F2;
 - _F4;
 - _F5;
 - _Enter;
 - _Ins;
 - gets-values;
 - openList;
 - openInsert;
 - openEdit.
- **URL адреса;**
- **Метод HTTP запиту;**
- **Параметри запиту (Назва-Значення);**
- **Параметри запиту (Назва-Тип даних);**
- **Модель прив'язки;**
- **Поля, що повертаються;**
- **Умова виконання події;**
- **Тег процесу;**
- параметр **Виконувати подію при старті модуля.**

Налаштування модуля. Вкладка **Події**.

Зліва міститься панель інструментів для створення графічного інтерфейсу, на якій знаходяться елементи інтерфейсу за розділами: **Назва шаблону, Докменеджер, Контроли, Реєстри, Графіки, Вкладення, Інше** (табл. 1).

Справа – область для розміщення елементів графічного інтерфейсу модуля. Для розміщення елементів графічного інтерфейсу спершу необхідно створити вкладку натиснувши кнопку **Додати вкладку** (). При створенні вкладки задаються реквізити вкладки:

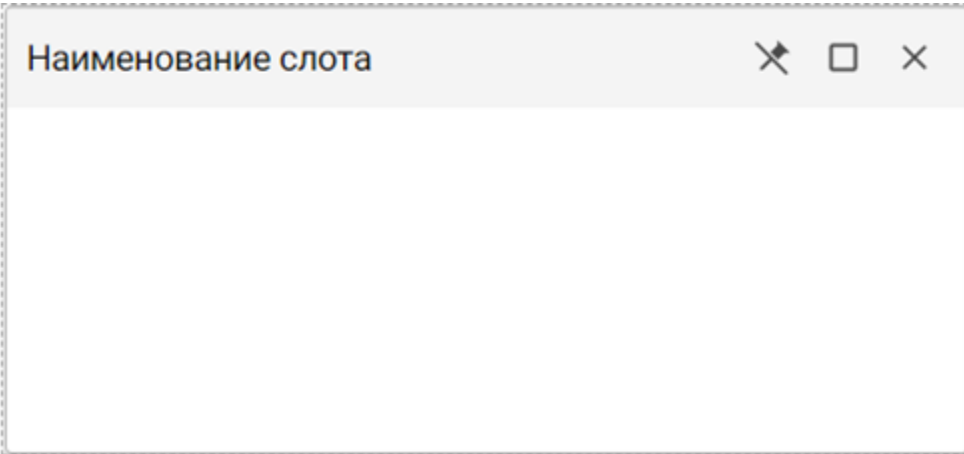
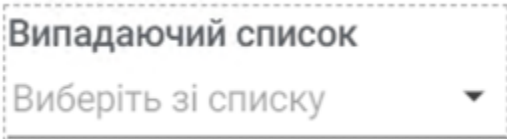
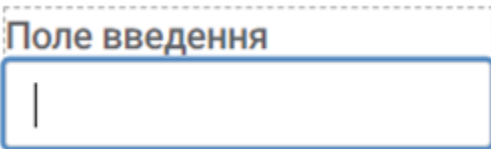
- **ID** вкладки;
- **Найменування** вкладки;
- **ID меню** вкладки.

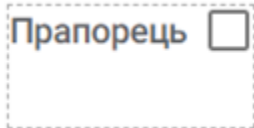
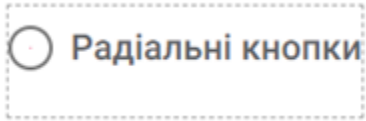
Для розміщення елементів графічного інтерфейсу необхідно обрати елемент з панелі інструментів і методом перетягування перетягнути в праву область, – область вкладки.

Усі розміщені елементи на вкладці поміщаються у **контейнер**.

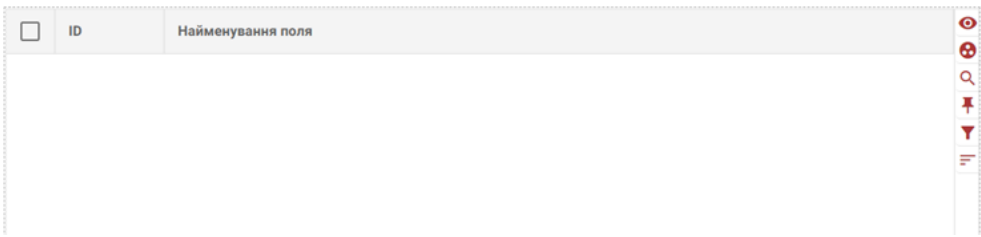
Для згортання / розгортання розділів панелі інструментів можна скористатися кнопками: – згорнути та – розгорнути.

Таблиця 1 – Панель інструментів Конструктора модулів

Графічний елемент інтерфейсу	Зовнішній вигляд
Розділ Докменеджер	
Спліт панель 	
Розділ Контроли	
Випадаючий список 	
Поле введення 	
Час 	
Графічний елемент інтерфейсу	Зовнішній вигляд
Дата 	

Графічний елемент інтерфейсу	Зовнішній вигляд
Дата 	
Період по роках 	
Дата та час 	
Прапорець 	
Радіальні кнопки 	
Текстове поле 	

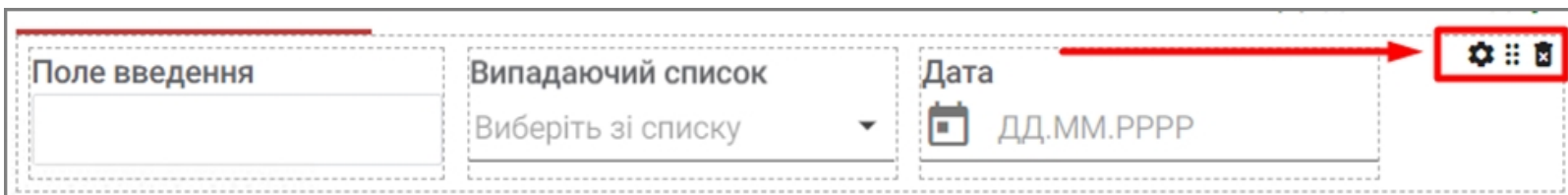
Розділ Реєстри

Реєстр 	
Графічний елемент інтерфейсу	Зовнішній вигляд
Каскадний реєстр 	

Графічний елемент інтерфейсу	Зовнішній вигляд																		
Каскадний реєстр																			
Ієрархічний реєстр																			
Розділ Графіки																			
Графік	Графік <table><caption>Дані для графіка</caption><thead><tr><th>Демо</th><th>Пурпурна лінія</th><th>Зелена лінія</th></tr></thead><tbody><tr><td>Демо</td><td>28000</td><td>18000</td></tr><tr><td>Демо</td><td>33000</td><td>28000</td></tr><tr><td>Демо</td><td>38000</td><td>8000</td></tr><tr><td>Демо</td><td>48000</td><td>23000</td></tr><tr><td>Демо</td><td>23000</td><td>33000</td></tr></tbody></table>	Демо	Пурпурна лінія	Зелена лінія	Демо	28000	18000	Демо	33000	28000	Демо	38000	8000	Демо	48000	23000	Демо	23000	33000
Демо	Пурпурна лінія	Зелена лінія																	
Демо	28000	18000																	
Демо	33000	28000																	
Демо	38000	8000																	
Демо	48000	23000																	
Демо	23000	33000																	
Розділ Вкладення																			
Вкладення	ID Найменування поля 📎 📎 📎 ✖ 🔍																		
Розділ Інше																			
Картка	⚙️ ❌ Назва поля ФІО ⚙️ ❌ Назва поля Посада (професія) ⚙️ ❌ Назва поля Микунець Михайло Юрій ⚙️ ❌ Назва поля Водій автотранспортних ⚙️ ❌ Назва поля Підрозділ ⚙️ ❌ Назва поля Додати поле + ⚙️ ❌ Назва поля Автотранспортний відділ																		

1.2.1. Налаштування контейнера графічних елементів

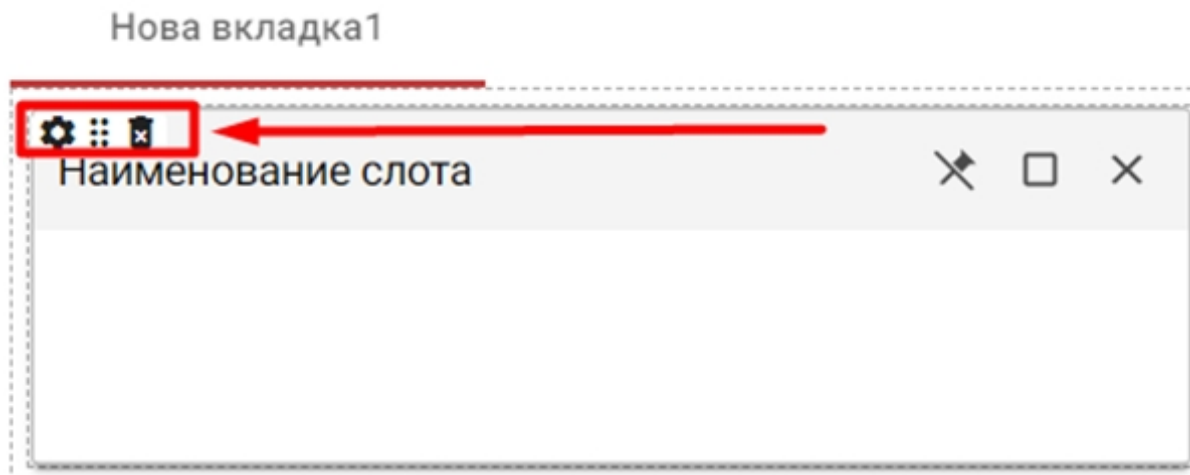
Розміщений на вкладці графічний елемент поміщається у **Контейнер**, який візуально групує всі елементи. При наведенні курсора мишки на **Контейнер**, у його верхньому правому куті відображаються кнопки:  – **Налаштувати контейнер**,  – **Перемістити контейнер**,  – **Видалити контейнер**.



Налаштування контейнера графічних елементів

1.2.2. Налаштування графічних елементів інтерфейсу

Розміщений у робочій області вкладки графічний елемент можна: налаштувати – , переміщувати –  або видалити –  за допомогою кнопок, що з'являються у лівому верхньому куті при наведенні на елемент.



Налаштування графічних елементів

Кожен графічний елемент інтерфейсу має свої параметри для налаштування. Налаштування елемента можна виконати при його розміщенні в робочій області та після його розміщення натиснувши на кнопку.

Вікно налаштування містить кнопки у правому верхньому куті, що дозволяють:

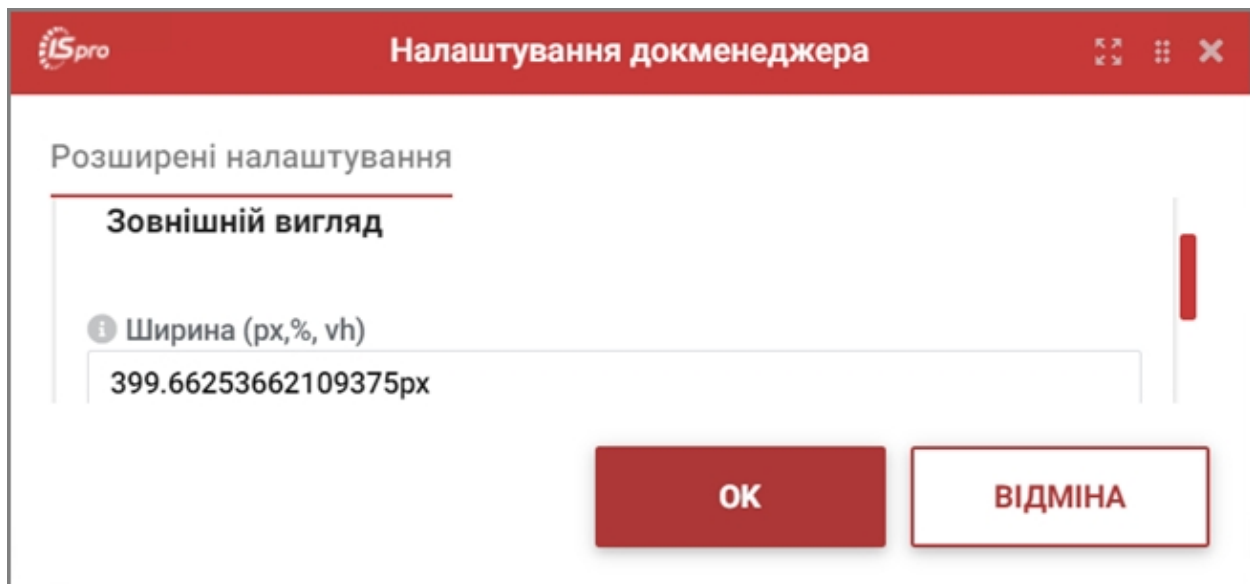
 – збільшити вікно налаштування;

 – перемістити вікно налаштування;

 – закрити вікно налаштувань.

По кнопці **ОК** зберігаються встановлені налаштування для елемента графічного інтерфейсу.

По кнопці **Відміна** скасовуються встановлені налаштування і закривається вікно налаштування.



Вікно **Налаштування графічних елементів**

1.2.3. Налаштування елемента Спліт панель

Налаштування **Спліт панелі** здійснюється по кнопці – . Для **Спліт панелі** доступні налаштування **Зовнішнього вигляду**:

- **Ширина** (px, %, vh);
- **Висота** (px, %, vh).

iSpro Налаштування докменеджера

Розширені налаштування

Зовнішній вигляд

❶ Ширина (px, %, vh)
399.66253662109375px

❶ Висота (px, %, vh)
100%

ОК ВІДМІНА

Налаштування Спліт панелі

1.2.4. Налаштування елемента Випадаючий список

Налаштування **Випадаючого списку** здійснюється по кнопці – . Для **Випадаючого списку** доступні налаштування, розміщені на вкладках: **Основні налаштування**, **Події**, **Розширені налаштування**.

На вкладці **Основні налаштування** містяться поля:

- ID;
- Найменування;
- Значення за замовчуванням;
- Умова активності поля;
- Умова приховування поля;
- Умова тільки для читання поля;
- Умова обов'язковості поля;
- Тип значення – обирається зі списку:
 - Текст;
 - Ціле число;

- Дробове число;
- Дата;
- Час;
- Електронна пошта;
- Гіперпосилання.
- Унікальний ключ;
- Шаблон відображення записів у списку: Код: Назва або Назва;
- Умова відображення кнопки редагування;
- Умова відображення футера.

На вкладці **Події** створюються події для елемента по кнопці **Додати подію**.

На вкладці **Розширені налаштування** містяться розділи: **Залежності**, **Зв'язки з батьківським полем**, **Зовнішній вигляд**.

Створення залежності виконується по кнопці **Додати залежність**, де заповнюються поля:

- Подія;
- Код;
- Умова виконання залежності.

У розділі **Зв'язки з батьківським полем** відображаються зв'язки з батьківським полем.

У розділі **Зовнішній вигляд** вказується: **Ширина** (px, %, vh), **Висота** (px, %, vh).

1.2.5. Налаштування елемента Поле введення

Налаштування **Поле введення** здійснюється по кнопці – . Для **Поля введення** доступні налаштування, розміщені на вкладках: **Основні налаштування**, **Події**, **Розширені налаштування**.

На вкладці **Основні налаштування** містяться поля:

- ID;
- Найменування;
- Значення за замовчуванням;
- Умова активності поля;
- Умова приховування поля;
- Умова тільки для читання поля;
- Умова обов'язковості поля;

- **Тип значення** – обирається зі списку: **Текст, Ціле число, Дробове число, Дата, Час, Електронна пошта, Гіперпосилання.**
- **Маска.**

На вкладці **Події** створюються події для елемента по кнопці **Додати подію**.

На вкладці **Розширені налаштування** містяться розділи: **Залежності, Зв'язки з батьківським полем, Зовнішній вигляд.**

Створення залежності виконується по кнопці **Додати залежність**, де заповнюються поля:

- **Подія;**
- **Код;**
- **Умова виконання залежності.**

У розділі **Зв'язки з батьківським полем** відображаються зв'язки з батьківським полем.

У розділі **Зовнішній вигляд** вказується: **Ширина** (px, %, vh), **Висота** (px, %, vh).

1.2.6. Налаштування елементів: Час, Дата, Період по роках, Дата та час, Прапорець, Текстове поле

Налаштування елементів **Час, Дата, Період по роках, Дата та час, Прапорець, Текстове поле** здійснюється по кнопці – . Доступні налаштування, розміщені на вкладках: **Основні налаштування, Події, Розширені налаштування.**

На вкладці **Основні налаштування** містяться поля:

- **ID;**
- **Найменування;**
- **Значення за замовчуванням;**
- **Умова активності поля;**
- **Умова приховування поля;**
- **Умова тільки для читання поля;**
- **Умова обов'язковості поля;**
- **Тип значення** – обирається зі списку: **Текст, Ціле число, Дробове число, Дата, Час, Електронна пошта, Гіперпосилання.**

На вкладці **Події** створюються події для елемента по кнопці **Додати подію**.

На вкладці **Розширені налаштування** містяться розділи: **Залежності**, **Зв'язки з батьківським полем**, **Зовнішній вигляд**.

Створення залежності виконується по кнопці **Додати залежність** – , де заповнюються поля: **Подія**, **Код**, **Умова виконання залежності**.

У розділі **Зв'язки з батьківським полем** відображаються зв'язки з батьківським полем.

У розділі **Зовнішній вигляд** вказується: **Ширина** (px, %, vh), **Висота** (px, %, vh).

1.2.7. Налаштування елемента Радіальні кнопки

Налаштування **Радіальні кнопки** здійснюється по кнопці –  . Для **Радіальних кнопок** доступні налаштування, розміщені на вкладках: **Основні налаштування**, **Події**, **Розширені налаштування**.

На вкладці **Основні налаштування** містяться поля:

- ID;
- Найменування;
- Значення за замовчуванням;
- Умова активності поля;
- Умова приховування поля;
- Умова тільки для читання поля;
- Умова обов'язковості поля;
- Тип значення – обирається зі списку:
 - Текст;
 - Ціле число;
 - Дробове число;
 - Дата;
 - Час;
 - Електронна пошта;
 - Гіперпосилання.
- Орієнтація кнопок:
 - Вертикальна;
 - Горизонтальна.

На вкладці **Події** створюються події для елемента по кнопці **Додати подію**.

На вкладці **Розширені налаштування** містяться розділи: **Залежності**, **Зв'язки з батьківським полем**, **Зовнішній вигляд**.

Створення залежності виконується по кнопці **Додати залежність**, де заповнюються поля:

- Подія;
- Код;
- Умова виконання залежності.

У розділі **Зв'язки з батьківським полем** відображаються зв'язки з батьківським полем.

У розділі **Зовнішній вигляд** вказується: **Ширина** (px, %, vh), **Висота** (px, %, vh).

1.2.8. Налаштування елемента Ієрархічний реєстр

Налаштування **Реєстру, Ієрархічного реєстру** здійснюється по кнопці – . Доступні налаштування, розміщені на вкладках: **Основні налаштування, Події, Параметри, Розширені налаштування**.

На вкладці **Основні налаштування** містяться поля:

- ID;
- Найменування;
- ID модуль;
- Register ID;
- Query ID;
- Унікальний ID запису – обирається значення зі списку;
- Значення за замовчуванням;
- Серверна відмітка.

На вкладці **Події** створюються події для елемента по кнопці **Додати подію**.

На вкладці **Параметри** відображаються поля: **ID, Тип, Значення**.

На вкладці **Розширені налаштування** містяться розділи: **Теги, Зовнішній вигляд**.

Створення тегу виконується по кнопці **Додати тег**, де заповнюються поля: **ID, Умова додавання**.

У розділі **Зовнішній вигляд** вказується: **Ширина** (px, %, vh), **Висота** (px, %, vh).

1.2.9. Налаштування елемента Каскадний реєстр

Налаштування **Реєстру** здійснюється по кнопці – . Для **Реєстру** доступні налаштування, розміщені на вкладках: **Основні налаштування, Події, Параметри, Розширені налаштування**.

На вкладці **Основні налаштування** містяться поля:

- **ID;**
- **Найменування;**
- **ID модуль;**
- **Register ID;**
- **Query ID;**
- **Унікальний ID запису** – обирається значення зі списку;
- **Значення за замовчуванням;**
- **Серверна відмітка.**

На вкладці **Події** створюються події для елемента по кнопці **Додати подію**.

На вкладці **Параметри** відображаються поля: **ID, Тип, Значення**.

На вкладці **Розширені налаштування** міститься розділ: **Зовнішній вигляд**. У даному розділі вказується: **Ширина** (px, %, vh), **Висота** (px, %, vh).

1.2.10. Налаштування елемента Графік

Налаштування **Графіку** здійснюється по кнопці – . Для **Графіку** доступні налаштування, розміщені на вкладках: **Основні налаштування та Розширені налаштування**.

На вкладці **Основні налаштування** містяться поля:

- **ID;**
- **Найменування;**
- **Тип графіка** – обирається значення зі списку: Лінійний графік, Діаграма з областями, Стопчаста діаграма, Точкова діаграма, Сплайн-діаграма, Сплайн-діаграма з областями, Ступінчаста діаграма з областями;
- Джерело даних для графіка: **Реєстр** або **Події**;
- **Налаштувати реєстр** – відкривається вікно налаштувань для реєстру;
- Ключ запису за віссю X – обирається значення зі списку;
- Ключ запису за віссю Y – обирається значення зі списку;
- Заголовок вісі;

На вкладці **Розширені налаштування** містяться розділи: **Зовнішній вигляд**. У даному розділі вказується: **Ширина** (px, %, vh), **Висота** (px, %, vh).

1.2.11. Налаштування елемента Вкладення

Налаштування **Вкладення** здійснюється по кнопці – . Для **Вкладення** доступні налаштування, розміщені на вкладках: **Основні налаштування**, **Події** та **Розширені налаштування**.

На вкладці **Основні налаштування** містяться поля:

- ID;
- Найменування;

На вкладці **Події** створюються події для елемента по кнопці **Додати подію**.

На вкладці **Розширені налаштування** містяться розділи: **Зовнішній вигляд**. У даному розділі вказується: **Ширина** (px, %, vh), **Висота** (px, %, vh).

1.2.12. Налаштування елемента Картка

Налаштування **Картка** здійснюється по кнопці – . Для **Картки** доступні налаштування, розміщені на вкладках: **Основні налаштування**, **Події** та **Розширені налаштування**.

На вкладці **Основні налаштування** містяться поля:

- ID;
- Найменування;
- параметр **Відображати інформацію**;
- параметр **Відображати фото**.

На вкладці **Події** створюються події для елемента по кнопці **Додати подію**.

На вкладці **Розширені налаштування** містяться розділи: **Зовнішній вигляд**. У даному розділі вказується: **Ширина** (px, %, vh), **Висота** (px, %, vh).

Картка містить всередині елементи фото та поля. По кнопці **Додати поле** додаються необхідні поля до картки.

Налаштування **фото** здійснюється по кнопці – , де обирається **Формат фото**: Квадратна, Кругла, Користувачька. При виборі формату **Користувачька** вказується **Ширина** та **Висота** області для фото.

Налаштування **поля** здійснюється по кнопці – , де обирається **Колір тексту** (default, ispro, blue) та **Розмір тексту** (маленький, звичайний, великий). При виборі відповідних параметрів змінюється **Приклад тексту**.

Деякі поля можна по кнопці –  **Сховати / Показати** (). Кнопкою –  видаляється поле з контейнера.

1.3. API

API ISpro (Application Programming Interface) – програмний інтерфейс **Комплексу ISpro**, що представляє собою набір інструментів (структур, процедур, класів, функцій), властивостей і методів для взаємодії з об'єктами **Комплексу**.

REST API ISpro (Representational state transfer) – інструмент, що забезпечує взаємодію та обмін інформацією з різних джерел (виконує роль посередника): бази даних **Комплексу**, веб-системи **ISpro**, мобільних додатків та ін.

Обмін даними між різними системами (веб-системою, десктопною версією **Комплексу**, мобільними додатками) у **REST API** здійснюється за протоколом **HTTP**. У свою чергу, **REST API ISpro** надає список URL-адрес, за допомогою яких сервер аплікацій може приймати запити на отримання, збереження, оновлення і видалення даних.

Для опису **REST API** використано специфікацію **OpenAPI** (формальна назва – **Swagger**).

У **OpenAPI** замість **XML**-формату використовується набір **JSON**-об'єктів з певною схемою, яка визначає їх найменування, порядок і вміст. Ці файли **JSON**, часто описуються в **YAML** (**JSON**-подібна мова розмітки, з розширеними можливостями) і містять представлення кожної частини **API**.

Перелік актуальних доступних методів та контролерів наведено у сервері застосунків.



Сервер Станції Ліцензії Блокування Завантажити тонкий клієнт **ISpro Core API** Вихід

Перелік ліцензій

В правому верхньому куті по пункту **ISpro Core API** відкривається вікно переліку методів та контролерів. По кнопці **Пошук** можна знайти необхідний.



ISpro Core API

Сервер застосунків ISpro Core

Кількість контролерів: 662

Кількість методів: 5539

Пошук

TransferPrepaymentDBF XmlEngine

AccessAccountingObject Управление доступами к объектам учета.

AccessDocument Управление доступами к документам.

AccessJournal Управление доступами к журналам.

AccessPoint Точка доступа к справочникам

Account Логистика. Закупки/Сбыт. Счет к получению/Счет к оплате

1.3.1. Налаштування API

1.3.1.1. Підключення до ISpro Core API

Підключення до **ISpro Core API** здійснюється у форматі:

http:// <адреса сервера аплікацій>:<порт>/swagger

Адреса та порт сервера аплікацій задаються при встановленні сервера аплікацій, наприклад, –
http://127.0.0.1:12001/swagger

ISpro Core API має вбудований графічний інтерфейс, за допомогою якого оброблюються запити.

Після завантаження веб-сторінки відобразиться список запитів.

У полі **Пошук (Filter by tag)** задаються критерії пошуку запитів.

За допомогою URL-адрес виконується звернення до сервера аплікацій для виконання дій (основних методів).

Інформація з пов'язаних питань

Підключення API для роботи з Комплексом *(на стор. 98)*

1.4. Інструментарій програміста

В **Комплексі** для створення нестандартних проводок (проводок, які формуються за переліком умов), автоматичне формування текстових документів або коментарів (тексти наказів, договорів), розрахунку законодавчо невизначених процесів (наприклад, обробка банківських документів з програми клієнт–банк) та інших подій використовується програми користувача. Для створення програм використовується вбудована алгоритмічна мова. Для формування звітів необхідних для роботи користувача, крім представлених в **Комплексі**, існують засоби створення і формування вихідних форм і засоби доступу до об'єктів даних **Комплексу**.

1.4.1. Вбудований Редактор програм

Одна з найбільш важливих функцій **Комплексу** – можливість завдання алгоритмічних процедур. За допомогою вбудованих або програмованих користувачем процедур, що застосовуються в усіх системах **Комплексу**, відбудеться налаштування проводок по рахунках бухгалтерського обліку, алгоритмів розрахунку заробітної плати, податків, амортизації основних засобів, програмування і обчислення параметрів в звітах, настройка користувальницького фільтра, а також вирішуються інші завдання. Для написання процедур в комплекс вбудований **Редактор програм**, забезпечений довідником змінних, функцій і конструкцій мови (**Елементи програми**), а також функцією синтаксичного перевірки написаної процедури.

Інформація з пов'язаних питань

[Редактор програм \(на стор. 100\)](#)

[Елементи програми \(на стор. 101\)](#)

1.4.1.1. Редактор програм

Редактор програм викликається в тих місцях **Комплексу**, де потрібно налаштування алгоритму програмним способом. Наприклад, в підсистемі **Головна книга** редактор програм викликається з розділу **Проведення** модуля **Налаштування** для додавання або зміни алгоритму формування проводки.

Щоб змінити задану проводку, курсор встановлюється на потрібний пункт лівій панелі реєстру і потім по клавіші **Tab** курсор переводиться на праву панель. Далі вибирається спосіб завдання проводки **Програмний** і по клавіші **F4** викликається **Редактор програм**.

Редактор програм це текстовий редактор, що підтримує стандартні функції редагування тексту – виділення фрагмента, видалення фрагмента, копіювання в буфер обміну, вставка з буфера. Виділення фрагмента тексту виконується по комбінації клавіш **Shift** і управління курсором. Знімається виділення натисканням будь-якої клавіші. Команди вставити, видалити, вирізати і копіювати виконуються через пункт меню **Правка**. Вставка проводиться з буфера обміну або зі спеціального довідника **Елементи програми**. Пункт меню **Правка / Знайти** дозволяє знайти в тексті програми заданий рядок. Пункт меню **Правка / Знайти далі** обробить решту тексту і знайде, де дана рядок зустрічається в наступний раз. Пункт меню **Правка / Зберегти блок в файл** збереже виділений фрагмент програми в існуючий текстовий файл, перезаписавши його вміст. В текст програми вставляється вміст текстового файлу, вставка виконується через пункт меню **Правка / Вставити блок з файлу**.

Перевірка правильності написання програми, виявлення помилок виконується по пункту меню **Програма / Перевірка**, або по комбінації клавіш **Ctrl + F9**.

Використання пошуку в **Редакторі програм**. У вікні редагування програми користувача в режимі пошуку (**F7**) реалізовано підсвічування всіх результатів пошуку в тексті програми.

При формуванні програми при відсутності полів є можливість активувати ресурси інших систем **Комплексу**. Для їх активації необхідно підключити відповідний ресурс по кнопці **Ресурси** у вікні **Елементи програми**. У вікні редагування тексту існує можливість коментування виділених рядків коду по комбінації клавіш **Ctrl + K**. Збереження програми відбувається по клавіші **F2**. При виході з редактора Система перевіряє текст програми і повідомляє про виявлені помилки. Перевірка виконується по комбінації клавіш **Ctrl + F9**, а друк документа по клавіші **F9**.

По клавіші **F3** (або меню **Правка / Викликати довідник**) з'являється вікно **Елементи програми** зі списком доступних елементів програми.

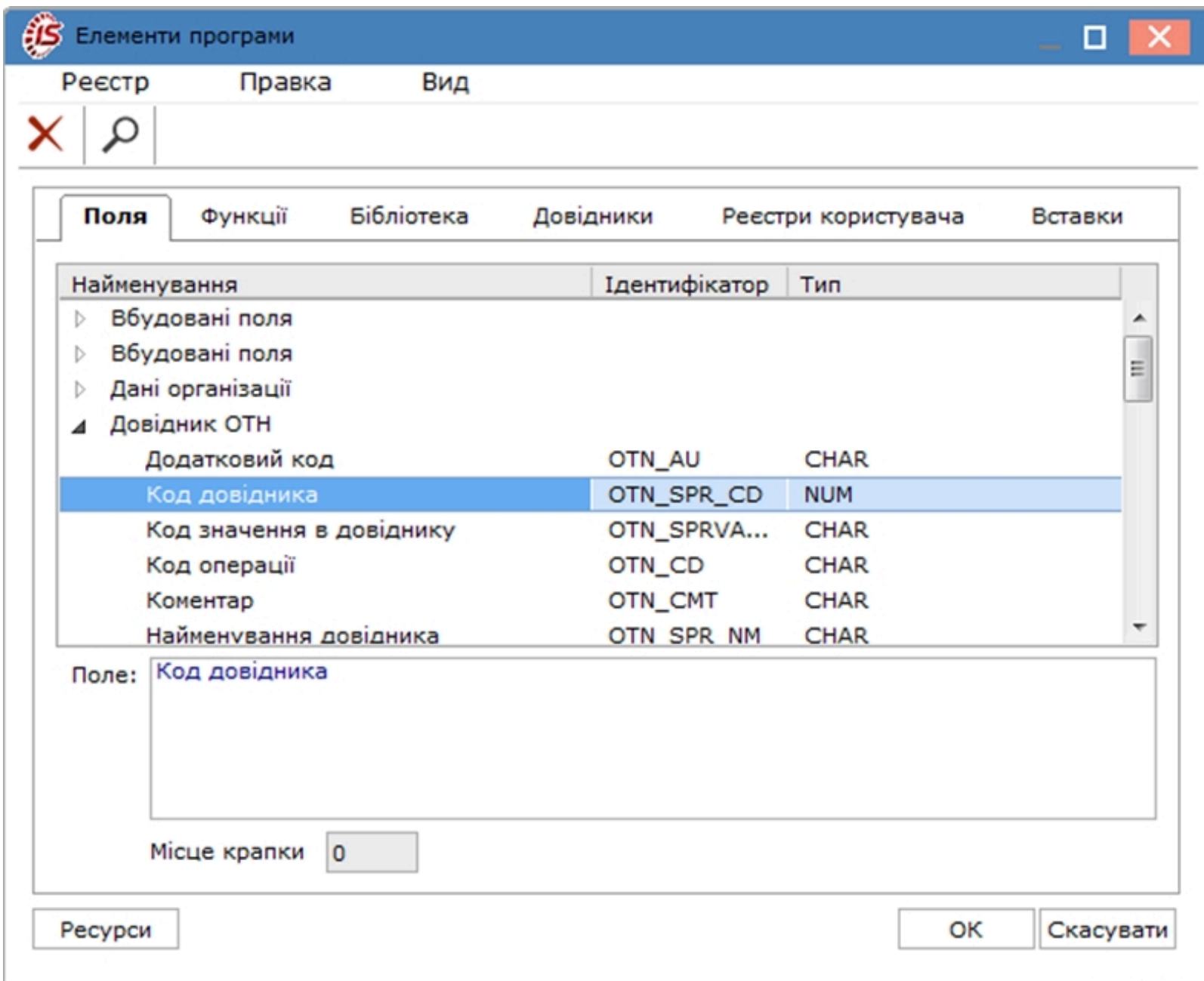
Для використання в редакторі програм полів і функцій, представлених в **Комплексі** по кнопці **Ресурси** підключають поля і функції інших систем.

1.4.1.2. Елементи програми

При написанні програм зручно використовувати вбудований довідник **Елементи програми**, що містить вкладки **Поля**, **Функції**, **Бібліотека**, **Довідники**, **Реєстри користувача**, **Вставки**. Довідник елементів програм викликається з пункту меню **Правка / Викликати довідник**, або по клавіші **F3**. При завантаженні модуля завантажуються дані, що відносяться до даної системи. У кожній системі відображаються загальні дані і дані системи. По кнопці **Ресурси** проводиться під'єднання полів і функцій інших систем.

Список елементів програми містить наступні вкладки:

- **Поля;**
- **Функції;**
- **Бібліотека;**
- **Довідники;**
- **Реєстри користувачів;**
- **Вставки.**



Вікно Елементи програми

1.4.1.2.1. Поля

Вкладка **Поля** містить список доступних параметрів і змінних, які використовуються при створенні програми, зокрема константи. Для зручності перегляду всі параметри розділені на кілька груп. Набір груп і параметрів, що входять в ці групи, залежить від системи і призначення настанованого алгоритму. Так, наприклад, група параметрів **Вбудовані поля** пропонується для

настройки проводок в різних системах. При налаштуванні різних форм звітності пропонуються групи **Системні поля**, **Стандартні поля**, **Реквізити партії**, **Реквізити фірм**, **Користувальницькі поля звіту**, **Аліасні поля звіту**, **Поля користувача** і ін. При цьому набір параметрів першої групи постійний, в другу групу включені параметри, які характерні тільки для даного виду звіту. В розділ **Користувальницькі поля** звіту можна самостійно ввести новий параметр. У цьому випадку буде потрібно визначити алгоритм розрахунку даного параметра.

Для кожного поля в довіднику показані **Найменування**, **Ідентифікатор** і **Тип**. Для написання алгоритмів використовується **Ідентифікатор**, текстове **Найменування** тільки пояснює призначення даного параметра. **Тип** поля може бути: **NUM** - числове, **CHAR** - символічне, **DATE** - дата, **TIME** - час. Необхідно стежити за тим, щоб у виразах брали участь параметри одного типу. При необхідності використовуються функції перетворення типів. Формат поля **DATE** – дата «**МІСЯЦЬ / ДЕНЬ / РІК**».

Довідник елементів програм містить досить велику кількість полів, тому важлива функція пошуку полів, що викликається по клавіші **F7**. Ви можете шукати за назвою, за ідентифікатором з урахуванням або без урахування регістру. Для пошуку необхідно встати на початок списку, так як пошук здійснюється один раз від поточного поля вниз по списку. Здійснивши пошук, курсор встановлюється на наступне поле після знайденого.

Можливо введення частини ідентифікатора, а відсутні символи замінюються зірочкою. Наприклад, для пошуку поля **Сума проводки**, вибирається спосіб пошуку **По найменуванню** та вказується в поле **Найменування: Сума ***. При цьому команда пошуку знайде перше поле, що містить зазначений набір - **Сума проводки**. Пошук всіх полів, що задовольняють вказаній вимозі, проводиться по комбінації клавіш **Alt + F7**. Таким чином, знаходяться послідовно поля **Сума операції**, **Сума комісії**.



Примітка:

Поточний перелік полів та функцій можна вивантажити до **Excel**. По кнопці **Ресурси** відмітити всі або необхідні системи та по пункту меню **Реєстр / Експорт списку** отримати перелік полів та функцій поточної версії Комплексу, які можна використати для написання програм.

1.4.1.2.2. Функції

Вкладка **Функції** містить список загальносистемних функцій і функцій для доступу до даних **Комплексу**. Загальносистемні функції доступні у всіх системах. Функції для полів користувача доступні у всіх системах. Комплекс містить вбудовані функції, які застосовуються для перетворення дат і валют, генерації проводок, розрахунку оборотів, балансу і аналізу по

аналітичним картками, тощо. У нижній частині розділу знаходиться підказка по функції, на якій встановлений курсор - формат функції і короткий коментар.

1.4.1.2.3. Бібліотека

Вкладка **Бібліотека** містить створені розробниками програми для виконання окремих операцій, які можна використовувати при написанні своїх програм. Дані програми представлені в розрізі систем.

Шаблон обраної функції (текст програми) відображається в нижній частині вікна і може бути вставлений в призначену для користувача програму натисканням на клавішу **Enter** або натисканням кнопки **OK**.

1.4.1.2.4. Довідники

Вкладка **Довідники** містить список системних, користувацьких і аналітичних довідників. Для вставки елемента в програму встановлюється в довіднику на нього курсор і натискається клавішу **Enter**.

1.4.1.2.5. Реєстри користувача

Вкладка **Реєстри користувача** містить реєстри користувача **Комплексу**.

1.4.1.2.6. Вставки

На вкладці **Вставки** здійснюється підключення своїх власних текстів бібліотек до програм типових операцій, калькуляцій та розрахункових таблиць аналогічно **#include file_name** з **C + +**. Файли з текстами бібліотек для підключення до програм зберігаються в серверної частини **Комплекту ISpro**, в каталозі **USER_DATA \ INCLUDE**. Файли мають розширення **VBA**.

1.4.2. Скрипти користувача ISpro WEB

У поля документа та модуля, додана можливість прописувати користувацькі скрипти на події елементів керування.

Мова написання скриптів: **JavaScript**.

У налаштуванні події існує блок **Скрипт користувача**, в якому можна ввести будь-який довільний скрипт, який буде виконаний за даною подією.

У цьому розділі є два блоки **До** та **Після**.

Блок **До**:

Виконується до самої події, тобто якщо в комбо є подія отримання значення для комбо і ви прописали скрипт в блок до, то він виконається до виклику методу сервера.

Блок Після:

Скрипт виконається після виконання події для отримання даних для комбо.

У скрипті на даний момент доступно 3 локальні змінні, які можна використовувати:

isproControls – Словник елементів керування, які є на формі.

isproModule - Об'єкт модуля, де описані параметри модуля, в тому числі елементи керування та модель (у випадку документа, у моделі немає моделі)

isproToast – об'єкт/сервіс, за допомогою якого можна вивести повідомлення користувачу.

Для приклада роботи скриптів можна подивитися 2 документи, один з них це в системі **Документообіг та управління процесами** в модулі **Документи**, шаблон документа **«Вхідні листи»**

У налаштування шаблону додано подію ініціалізації документа, в якому проставляємо, наприклад, базові значення полів.

Для роботи зі скриптами потрібно мати хоча б базове розуміння роботи такої мови, як JavaScript.

А саме оператори розгалуження, типи даних та їх поведінка залежно від контексту.

Масиви та методи роботи з ними.

Приклад №1. Заповнення поля

Складність: **Просто**

Відкрити налаштування шаблону:

ISpro Конструктор модулей Правка ? История

43

В СИСТЕМЕ Адм adm

Название шаблона
Входящее письмо

Добавить вкладку

Реквизиты

- Заместитель
- Процесс
- Дата
- Номер
- Автор
- Rdic файл отчета
- Автор версии
- Автор закрытия поручения
- Автор поручения
- Адрес кем получено
- Адрес отправителя
- Активная
- Аспект поступления
- Версия
- Вид
- Вид входящего
- Вид доверенности
- Вид документа
- Вид доставки
- Вид исходящего
- Вид нормативного докум...
- Вид обращения
- Файл документа

Номер

Дата
ДД.ММ.ГГГГ

Срок исполнения
ДД.ММ.ГГГГ

Индекс входящего

На №

От кого
Выберите из списка

Корреспондент
Выберите из списка

Краткое содержание

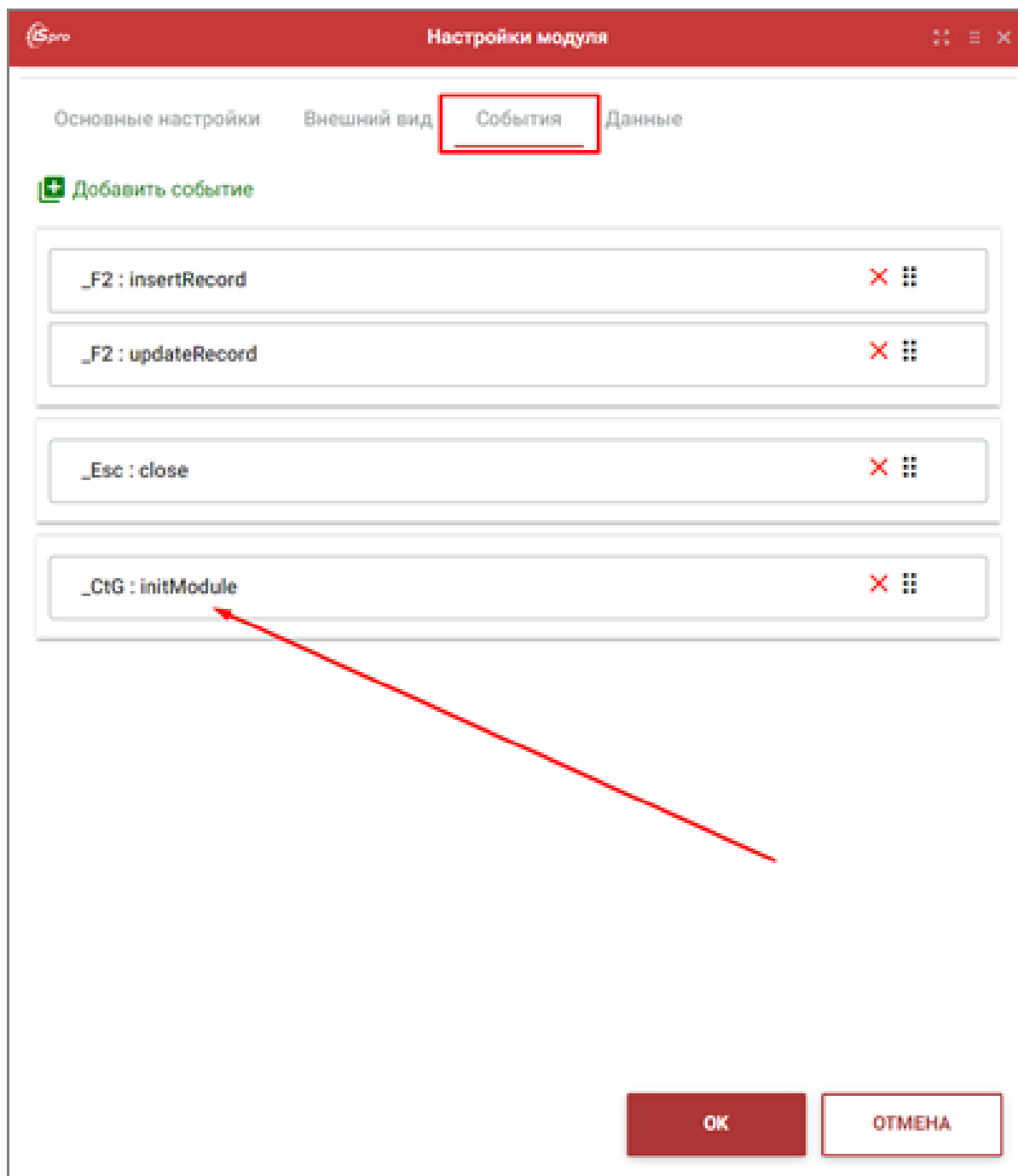
Исполнитель
Выберите из списка

Комментарий

Документооборот

ISpro: 8.0.006 | 999

Переходимо на вкладку **Події** та обираємо подію **InitModule**:



У події з'являється новий блок **Скрипт користувача**:

Настройки модуля

Основные настройки Внешний вид **События** Данные

_CtG : initModule

Идентификатор *
_CtG

Тип *
Событие

Действие *
designer.initModule

Пользовательский скрипт

Описание

URL адрес

Метод HTTP запроса

Параметры запроса (Название:Тип:Значение)

Модель привязки

ОК ОТМЕНА

При розкритті відображається редактор ДО і ПІСЛЯ, їх поведінка описана вище. У першому блоці написано простий скрипт, який при ініціалізації проставляє значення за деякими полями документа (Вхідний лист).



Текст скрипту:

```
isproControls.nmr = isproControls.nmr.value + 'Скрипт!';
```

```
isproControls.ssdcc = 'Це поле заповнене скриптом автоматично під час ініціалізації документа';
```

```
isproControls.na = '№555666777';
```

```
isproToast.showMessage('Скрипт', `Поля: ${isproControls.na.caption}, ${isproControls.ssdcc.caption},
${isproControls.nmr.caption} заповнені автоматично скриптом`, 1);
```

Спочатку потрібно зрозуміти яким чином набувати значення і надавати його.

isproControls в ньому знаходяться всі поля вікна, для отримання значення якого необхідно спочатку

визначити, яке поле цікавить.

Всі поля в даному об'єкті знаходяться під своїми id і ,наприклад, для отримання значення поля номер потрібно звернутися так : isproControls.nmr.value

Для надання значення достатньо звернеться isproControls.nmr = будь-яке значення.

У даному скрипті на 5 рядку, проводиться звернення до поля форми isproControls.nmr(Поле номер документа) і надається йому значення себе ж (isproControls.nmr.value) + текст «Скрипт!».

На шостому рядку присвоюємо значення поля **Коментар** - довільний текст.

На сьомому рядку аналогічно надають значення полю **На №** - довільний номер.

На восьмому рядку за допомогою об'єкта isproToast та його методу showMessage() виводиться користувачеві повідомлення, де першим параметром методу передається заголовок повідомлення, другим параметром текст повідомлення, останнім параметром тип повідомлення

Інформаційне повідомлення – 0,

Успішне повідомлення (зелена галочка) – 1,

Помилка – 2

Попередження – 3,

Приклад №2. Виконання умов.

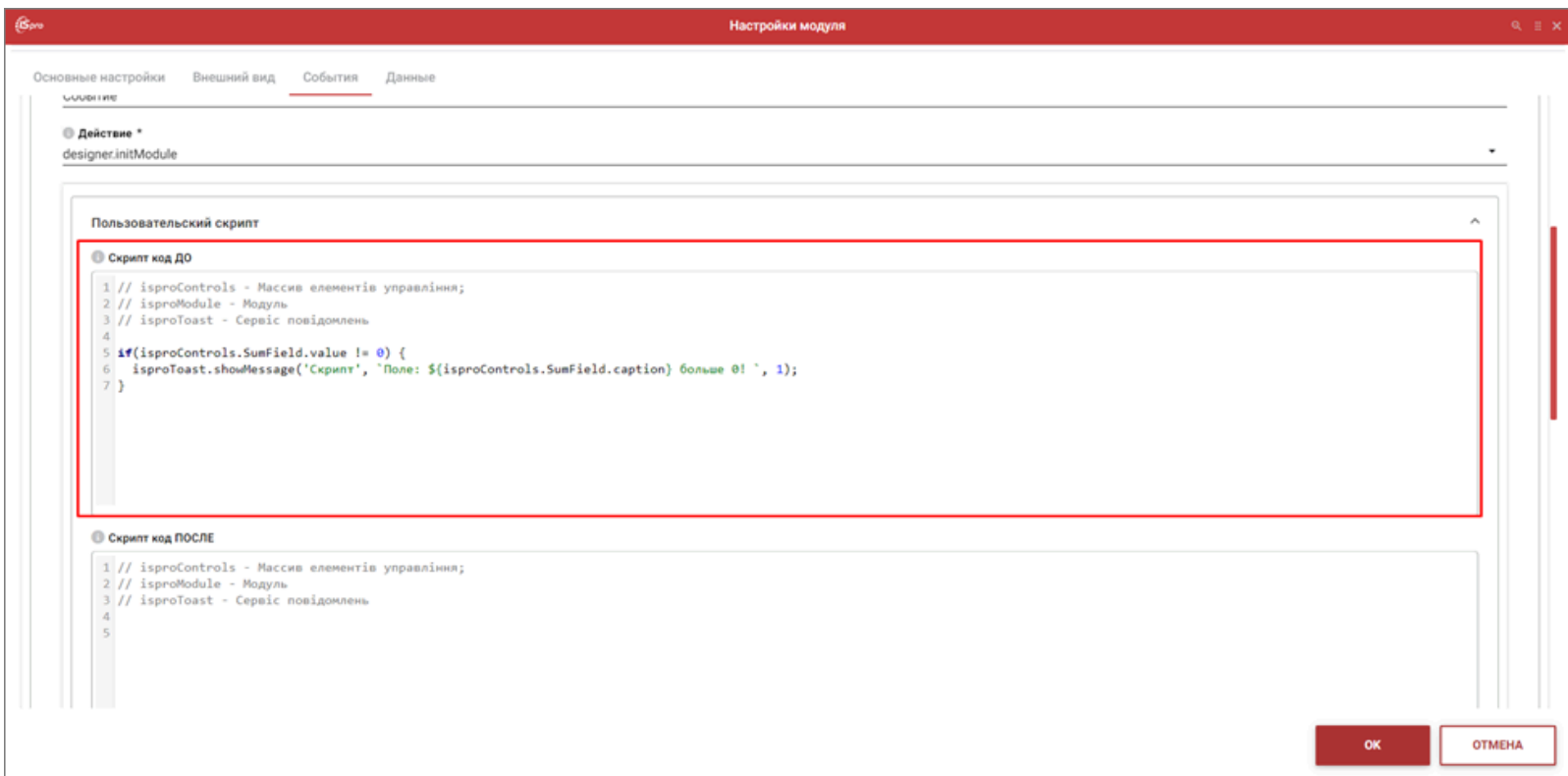
Складність: **Легко**

Приклад із операторами розгалуження.

Модуль: **Рахунки до оплати**

Документ: **Рахунки до оплати**

Ця подія розташована аналогічно першому прикладу.



Текст скрипту:

```
if(isproControls.SumField.value != 0) {isproToast.showMessage('Скрипт', `Поле:
${isproControls.SumField.caption} більше 0! `, 1);}
```

У цьому скрипті перевіряємо суму по полю, якщо вона не дорівнює 0, то в даному випадку буде виведено повідомлення інакше нічого не відбудеться.

Приклад №3. Складові значення полів

Складність: **Середня**

Модуль: **Документообіг та управління процесами/ Документи**

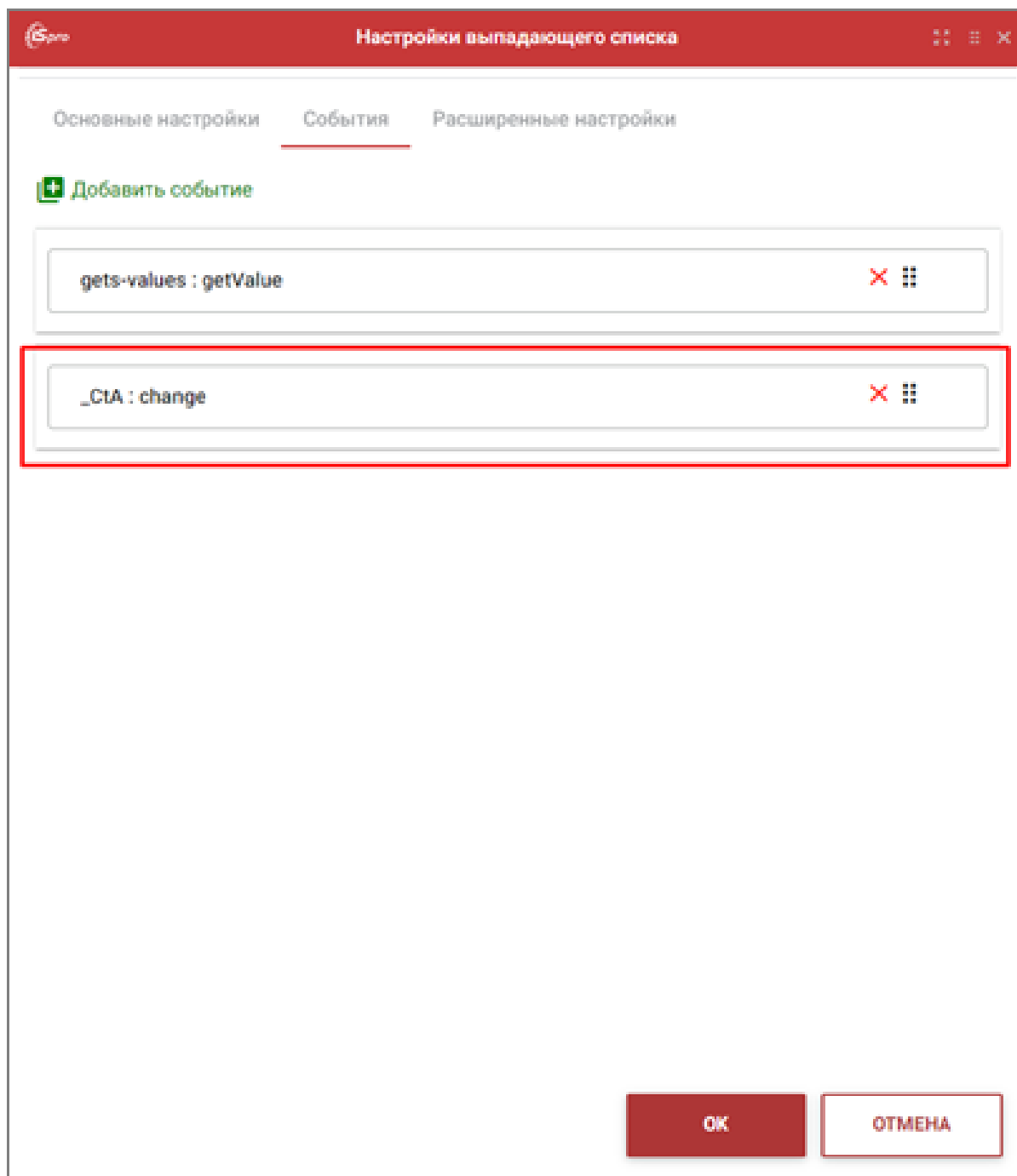
Шаблон: **Користувача Вхідний лист**(номер шаблону 4 код 1)

У цьому прикладі скрипт доданий у вибір **Від кого** на подію зміни значення комбо:

Відкриваємо параметри поля **Від кого**.

The screenshot displays the 'ISpro' software interface for configuring a document template. The top navigation bar includes 'Конструктор модулей', 'Правка', '?', and 'История'. The main area is titled 'Входящее письмо' and contains several input fields for document metadata. A sidebar on the left lists various 'Реквизиты' (Fields) such as 'Заместитель', 'Процесс', 'Дата', 'Номер', 'Автор', etc. A red box highlights the 'Подія' (Event) icon in the 'Реквизиты' list, with a red arrow pointing to it. The main form includes fields for 'Номер', 'Дата', 'Срок исполнения', 'Индекс входящего', 'На №', 'Корреспондент', 'Краткое содержание', 'Исполнитель', 'Вид нормативного документа', and 'Комментарий'.

Відкриваємо вкладку **Події**. Цікавить подія зміни значення.



Також відкриваємо налаштування події і дивимося в розділ **Скрипт користувача**:

Суть роботи скрипта в тому, що при зміні значення комбо **Від кого** необхідно сформувати префікс для поля номер документа.

Префікс формуватимемо за першими літерами ПІБ.

Приклад цього скрипту:

❶ Скрипт код ДО

```

5 let recipient = isproModule.model.ot.name;
6
7 if(isproControls.nmr.value == null || isproControls.nmr.value == '') {
8   isproControls.nmr = recipient.split(" ").map(s => s[0]).join("") + '-' + '';
9   isproToast.showMessage('Пользовательский скрипт №1', 'Скрипт отработал, первый блок', 1);
10 }else if (isproControls.nmr.value.includes('-')){
11   isproControls.nmr = isproControls.nmr.value.split('-')[1];
12   isproControls.nmr = recipient.split(" ").map(s => s[0]).join("") + '-' + isproControls.nmr.value;
13   console.log(isproControls);
14   isproControls.dat.isEnabled = 'false';
15   isproToast.showMessage('Пользовательский скрипт №2', 'Скрипт отработал, второй блок', 2);
16 }else{
17   isproControls.nmr = recipient.split(" ").map(s => s[0]).join("") + '-' + isproControls.nmr.value;
18   isproToast.showMessage('Пользовательский скрипт №3', 'Скрипт отработал, третий блок', 0);

```

Текст скрипту:

```
let recipient = isproModule.model.ot.name;
```

```
if(isproControls.nmr.value == null || isproControls.nmr.value == "") {
```

```
  isproControls.nmr = recipient.split(" ").map(s => s[0]).join("") + '-' + '';
```

```
  isproToast.showMessage('Скрипт користувача 1', 'Скрипт відпрацював, перший блок', 1);
```

```
}else if (isproControls.nmr.value.includes('-')){
```

```
  isproControls.nmr = isproControls.nmr.value.split('-')[1];
```

```
  isproControls.nmr = recipient.split(" ").map(s => s[0]).join("") + '-' + isproControls.nmr.value;
```

```
  console.log(isproControls);
```

```
  isproControls.dat.isEnabled = 'false';
```

```
  isproToast.showMessage('Скрипт користувача 2', 'Скрипт відпрацював, другий блок', 2);
```

```
}else{
```

```
  isproControls.nmr = recipient.split(" ").map(s => s[0]).join("") + '-' + isproControls.nmr.value;
```

```
  isproToast.showMessage('Скрипт користувача 3', 'Скрипт відпрацював, третій блок', 0);
```

}

Для його розуміння потрібно знати, як мінімум, методи роботи з масивами в мові JavaScript.

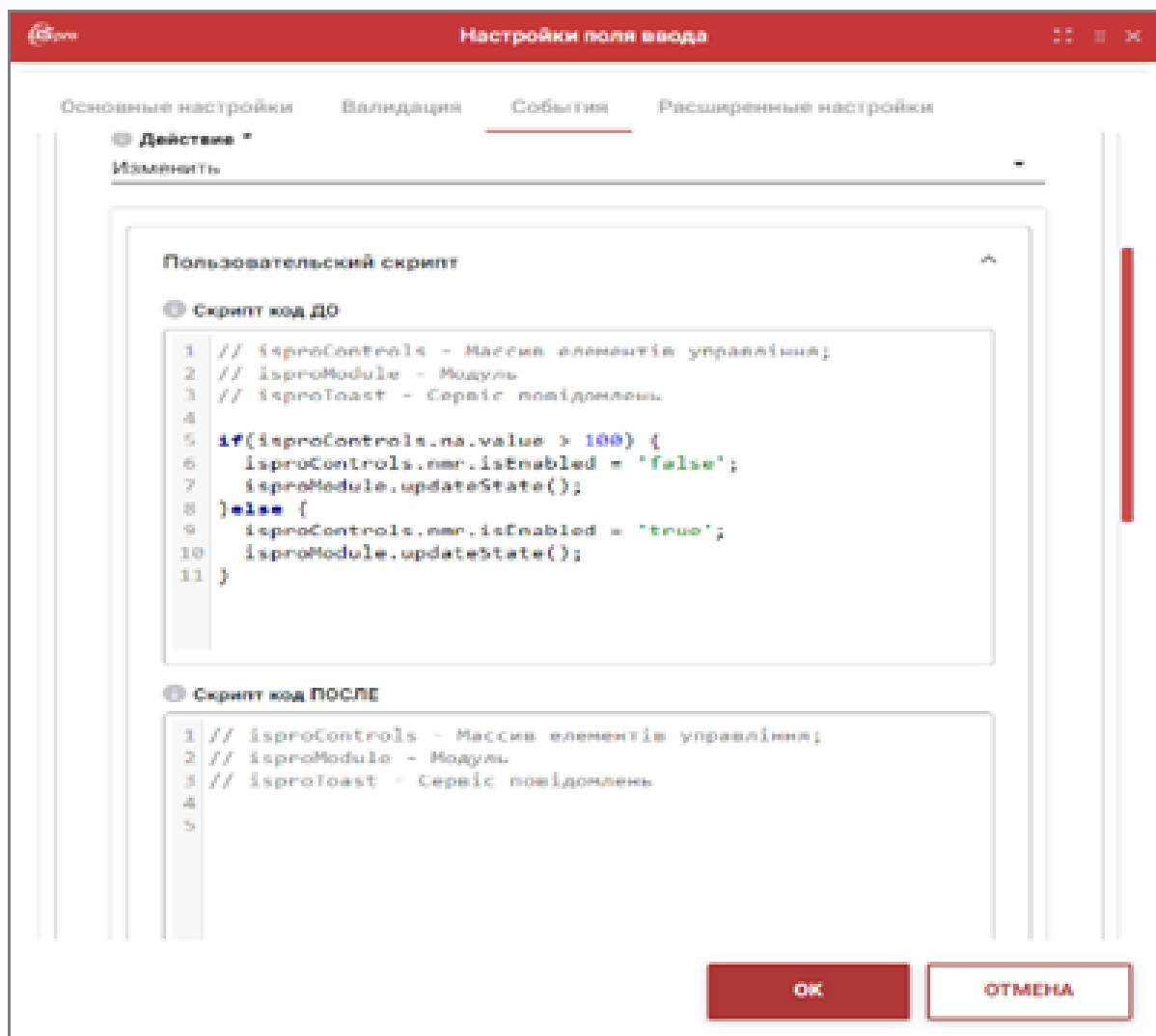
Приклад №4. Блокування полів

Складність: **Середня**

Модуль: **Документообіг та управління процесами / Документи**

Шаблон: **Користувальницький Вхідний лист системний шаблон.**

Поле **На №**, якщо значення даного поля буде більше 100, то поле номер буде заблоковано і якщо поле буде менше 100, то поле буде доступне для редагування.



Текст скрипта:

```

if(isproControls.na.value > 100) {

isproControls.nmr.isEnabled = 'false';

isproModule.updateState();

}else {

isproControls.nmr.isEnabled = 'true';

isproModule.updateState();

}

```

1.4.3. Дизайнер FastReport

1.4.3.1. Етапи побудови звіту

Виділяють наступні компоненти процесу побудови типового звіту:



Етапи створення звіту

Являють собою сукупність даних, на основі яких будується звіт. Організація доступу до даних з БД здійснюється компонентами **FastReport** без участі програміста.

Крім даних, що зберігаються в БД, **FastReport** використовує будь-які джерела (масив, файл, вміст **StringGrid** і ін.). В цьому випадку програміст сам піклуватися про доступ до такої інформації. Для цього використовується набір подій, що дозволяють здійснити передачу даних в компоненти **FastReport**.

Крім доступу до даних, визначених у проєкті, **FastReport** дозволяє створювати нові компоненти в реальному часі. У **FastReport** принципи створення компонентів доступу до даних максимально наближені до тих, що використовуються в середовищі **Delphi**. Так само, як і в **Delphi**, на форму кладеться компонент, і в інспекторі об'єктів налаштовуються його властивості.

1.4.3.2. Параметри

На цьому етапі здійснюється запит параметрів у користувача (наприклад, діапазон дат, за яким необхідно вивести дані). Деякі звіти обходяться без цього етапу або використовують фіксовану установку параметрів (без запиту їх значень в діалозі).

У **FastReport** є можливість запиту параметрів, якщо звіт використовує дані з запиту (**Query**). Для діалогу з користувачем використовується діалогове вікно. Крім того, для запиту параметрів використовується форма, розроблена в середовищі **Delphi**. Правда, при необхідності будь-яких змін в логіці роботи доведеться перекомпілювати проект.

FastReport, крім цього, дозволяє користувачеві самому розробляти форму діалогу. Процес нагадує побудова форми в середовищі **Delphi**: є набір стандартних елементів управління, які розташовуються на формі діалогу і налаштовуються їх властивості. За допомогою вбудованої мови **FastReport** дозволяє реалізувати необхідну логіку роботи діалогу і передати введені значення ядру генератора.

Можливість створення власних діалогів дуже корисна - укупі з іншими можливостями (створення джерел даних, використання вбудованої мови) вона дозволяє створювати звіти, максимально відв'язаних від середовища **Delphi**. Це дозволяє створювати нові звіти і модифікувати існуючі без переписування або перекомпіляції проекту.

1.4.3.3. Форма

Форма звіту являє собою набір елементів, що описують вид готового звіту. Для угруповання елементів по їх розташуванню в готовому звіті застосовуються бенди (англ. Band - смужка). Бенди поділяються на два види: службові (заголовок звіту, сторінки та ін.) і бенди, що утворюють багаторядкова частина звіту (далі - дата-бенди). Дата-бенди підключаються до джерел даних, і їх вміст виводиться стільки разів, скільки є рядків даних в джерелі.

Для побудови форми звіту використовується візуальне середовище розробки - дизайнер / Інтерфейс дизайнера виконаний на сучасному рівні з використанням панелей інструментів (**toolbars**), розташування яких змінюється. Для зручності маніпуляції властивостями об'єктів звіту використовується інспектор об'єктів, аналогічний делфійському.

1.4.3.4. Обробка

Під обробкою розуміється обробка вхідних даних, модифікація форми звіту або окремих її компонентів в процесі побудови звіту. Найпростіший приклад такої обробки - висновок від'ємних сум червоним кольором. Більш складний приклад обробки - друк суми, яка підраховується в підвалі групи, в її заголовку.

Реалізувати подібну обробку можна, вдавшись до написання обробників подій в **Delphi** - саме так і зроблено в **FastReport**. Цей спосіб не є універсальним, тому що не дозволяє створювати нові звіти поза середовищем Delphi без переписування і перекомпіляції проекту. Саме тому в **FastReport** застосована вбудована мова - спрощений аналог Pascal. Скрипти, написані на цій мові, по суті справи, є обробниками подій, що викликаються перед промальовуванням об'єктів. Це дає можливість виконувати досить складну обробку інформації без написання коду в **Delphi**, і відповідно, без жорсткої прив'язки звіту до проекту.

Можливості вбудованої мови **FastReport** досить широкі. З скрипта доступні всі властивості і методи об'єктів звіту, а також змінні, поля таблиць БД. В скрипті створюються змінні і масиви, які доступні в усьому звіті. Про можливості вбудованої мови говорить той факт, що така досить складна задача, як друк сум групи в її заголовку (сама сума вважається в підвалі групи) засобами мови **FastReport** робиться просто.

1.4.3.5. Готовий звіт

Готовий звіт являє собою продукт діяльності ядра **FastReport** - то, що користувач бачить при натисканні кнопки **Попередній перегляд**. На відміну від багатьох генераторів звітів, які зберігають вміст сторінок звіту у вигляді метафайлу (тобто зображення у форматі **EMF**), в **FastReport** готовий звіт представлений набором об'єктів, що описують вміст кожної сторінки звіту. Це дозволяє модифікувати готовий звіт, завантажуючи потрібну сторінку в дизайнер. Крім того, можна описувати реакцію на клацання миші на необхідному об'єкті в режимі попереднього перегляду звіту. Це дозволяє легко організувати роботу програми, при якій клацання на об'єкті звіту викликає генерацію нового звіту з більш детальними відомостями по вибраному об'єкту.

Вікно попереднього перегляду **FastReport** відрізняється від використовуваних в інших генераторах. Зокрема, в **FastReport** використовується схема перегляду документа і навігації, прийнята в **Microsoft Word**: у вікні можна бачити відразу кілька сторінок. Крім того, реалізована можливість пошуку тексту у всьому документі.

Документацію по FastReport можна знайти на сайті розробника.

1.4.3.6. Етапи побудови звіту

Виділяють наступні компоненти процесу побудови типового звіту:



Етапи створення звіту

1.4.3.6.1. Дані

Являють собою сукупність даних, на основі яких будується звіт. Організація доступу до даних з БД здійснюється компонентами **FastReport** без участі програміста.

Крім даних, що зберігаються в БД, **FastReport** використовує будь-які джерела (масив, файл, вміст StringGrid і ін.). В цьому випадку програміст сам піклуватися про доступ до такої інформації. Для цього використовується набір подій, що дозволяють здійснити передачу даних в компоненти **FastReport**.

Крім доступу до даних, визначених у проєкті, **FastReport** дозволяє створювати нові компоненти в реальному часі. У **FastReport** принципи створення компонентів доступу до даних максимально наближені до тих, що використовуються в середовищі Delphi. Так само, як і в Delphi, на форму кладеться компонент, і в інспекторі об'єктів налаштовуються його властивості.

1.4.3.6.2. Параметри

На цьому етапі здійснюється запит параметрів у користувача (наприклад, діапазон дат, за яким необхідно вивести дані). Деякі звіти обходяться без цього етапу або використовують фіксовану установку параметрів (без запиту їх значень в діалозі).

У **FastReport** є можливість запиту параметрів, якщо звіт використовує дані з запиту (**Query**). Для діалогу з користувачем використовується діалогове вікно. Крім того, для запиту параметрів використовується форма, розроблена в середовищі Delphi. Правда, при необхідності будь-яких змін в логіці роботи доведеться перекомпілювати проєкт.

FastReport, крім цього, дозволяє користувачеві самому розробляти форму діалогу. Процес нагадує побудова форми в середовищі Delphi: є набір стандартних елементів управління, які розташовуються на формі діалогу і налаштовуються їх властивості. За допомогою вбудованої мови **FastReport** дозволяє реалізувати необхідну логіку роботи діалогу і передати введені значення ядру генератора.

Можливість створення власних діалогів дуже корисна - укупі з іншими можливостями (створення джерел даних, використання вбудованої мови) вона дозволяє створювати звіти, максимально відв'язаних від середовища Delphi. Це дозволяє створювати нові звіти і модифікувати існуючі без переписування або перекомпіляції проекту.

1.4.3.6.3. Форма

Форма звіту являє собою набір елементів, що описують вид готового звіту. Для угруповання елементів по їх розташуванню в готовому звіті застосовуються бенди (англ. Band - смужка). Бенди поділяються на два види: службові (заголовок звіту, сторінки та ін.) і бенди, що утворюють багаторядкова частина звіту (далі - дата-бенди). Дата-бенди підключаються до джерел даних, і їх вміст виводиться стільки разів, скільки є рядків даних в джерелі.

Для побудови форми звіту використовується візуальне середовище розробки - дизайнер / Інтерфейс дизайнера виконаний на сучасному рівні з використанням панелей інструментів (**toolbars**), розташування яких змінюється. Для зручності маніпуляції властивостями об'єктів звіту використовується інспектор об'єктів, аналогічний делфійському.

1.4.3.6.4. Обробка

Під обробкою розуміється обробка вхідних даних, модифікація форми звіту або окремих її компонентів в процесі побудови звіту. Найпростіший приклад такої обробки - висновок від'ємних сум червоним кольором. Більш складний приклад обробки - друк суми, яка підраховується в підвалі групи, в її заголовку.

Реалізувати подібну обробку можна, вдавшись до написання обробників подій в Delphi - саме так і зроблено в **FastReport**. Цей спосіб не є універсальним, тому що не дозволяє створювати нові звіти поза середовищем Delphi без переписування і перекомпіляції проекту. Саме тому в **FastReport** застосована вбудована мова - спрощений аналог Pascal. Скрипти, написані на цій мові, по суті справи, є обробниками подій, що викликаються перед промальовуванням об'єктів. Це дає можливість виконувати досить складну обробку інформації без написання коду в Delphi, і відповідно, без жорсткої прив'язки звіту до проекту.

Можливості вбудованої мови **FastReport** досить широкі. З скрипта доступні всі властивості і методи об'єктів звіту, а також змінні, поля таблиць БД. В скрипті створюються змінні і масиви, які доступні в усьому звіті. Про можливості вбудованої мови говорить той факт, що така досить складна задача, як друк сум групи в її заголовку (сама сума вважається в підвалі групи) засобами мови **FastReport** робиться просто.

1.4.3.6.5. Готовий звіт

Готовий звіт являє собою продукт діяльності ядра **FastReport** - то, що користувач бачить при натисканні кнопки **Попередній перегляд**. На відміну від багатьох генераторів звітів, які зберігають вміст сторінок звіту у вигляді метафайлу (тобто зображення у форматі **EMF**), в **FastReport** готовий звіт представлений набором об'єктів, що описують вміст кожної сторінки звіту. Це дозволяє модифікувати готовий звіт, завантажуючи потрібну сторінку в дизайнер. Крім того, можна описувати реакцію на клацання миші на необхідному об'єкті в режимі попереднього перегляду звіту. Це дозволяє легко організувати роботу програми, при якій клацання на об'єкті звіту викликає генерацію нового звіту з більш детальними відомостями по вибраному об'єкту.

Вікно попереднього перегляду **FastReport** відрізняється від використовуваних в інших генераторах. Зокрема, в **FastReport** використовується схема перегляду документа і навігації, прийнята в **Microsoft Word**: у вікні можна бачити відразу кілька сторінок. Крім того, реалізована можливість пошуку тексту у всьому документі.

1.4.3.6.6. Документація по FastReport

Документацію по **FastReport** можна знайти на сайті виробника.

1.4.3.6.7. Приклад внесення змін до звіту

Наприклад, потрібно внести зміни до звіту № 1807 «Дні народження» - додати до даних посада працівника.

1. Заходимо в модуль «**Особисті картки**». Вибираємо пункт меню **Реєстр / Друк** (клавіші **F9**). На екрані відкривається вікно " **Вибір вихідної форми** ".
2. У вікні " **Вибір вихідної форми** " вибираємо пункт меню **Реєстр / Меню** (комбінація клавіш **Shift + F4**).
3. У вікні « **Налаштування меню звітів** » знаходимо відповідну форму звіту № 1807 «**Дні народження**»:
 - a. Необхідно встановити курсор на звіт і вибрати пункт меню **Реєстр / Породити** (комбінація клавіша **Shift + Ins**).
 - b. У вікні « **Породження форми звіту** » внести найменування нового звіту, після чого натиснути кнопку « **ОК** ».
 - c. На екрані розгортається вікно редагування звіту:
 - Звіт формату -FR складається з декількох сторінок:
 - сторінка з програмним кодом (на рис. вкладка " **Код**");
 - сторінка з деталями (тето-поля), які друкуються (на рис. вкладка " **Page1**");

- сторінка, яка містить вікно запуску звіту і SQL-запит, який заповнює деталі структури (на рис. вкладка "**Page2**"); \
 - Переходимо на вкладку "**Page2**" на запит Query1 (в нижній частині вікна) і розкриваємо його для редагування натисканням клавіші **Enter**. Відкривається редактор SQL-запитів.
 - i. У SQL-запиті не вистачає поля з найменуванням посади, тому доповнюємо перелік полів (група **SELECT**) запиту, текстом «DOL.SPRD_NMFULL». Ім'я поля, яке необхідно додати визначається з бази даних (**Карти даних** (електронний опис бази даних) або опис бази в документі **Керівництво програміста**).
 - ii. Зберігаємо зміни запиту, натискаючи на кнопку «**ОК**» (вона має форму зеленої галочки в верхньому лівому кутку вікна редагування SQL-запиту).
 - iii. Переходимо на вкладку "**Page1**" в центральну частину вікна. У рядку **MasterDate1** в поле [**Query1. "SPRPDR_NM"**] натискаємо клавішу **Enter** для його редагування. Відкривається вікно «**Редактор тексту**».
 - iv. У вікні «**Редактор тексту**» ставимо курсор на першу позицію. Натискаємо на кнопку «**Вставити вираз**». Зі списку виразів вибираємо поле **SPRD_NMFULL** і двічі натискаємо на ньому мишею. Натискаємо «**ОК**». Зберігаємо зміни у вікні «**Вставити вираз**», натискаючи на кнопку «**ОК**» (вона має форму зеленої галочки в верхньому лівому кутку вікна редагування).
 - v. При необхідності в рядку **PageHeader1** можна змінити назву стовпця, в якому додавали поле. Наприклад, змінюємо назву **Підрозділ** на **Підрозділ і посаду**.
 - vi. Зберігаємо зміни в звіті «**Зберегти**» (комбінація клавіша **Ctrl + S** або натиснути на малюнок з дискетою).
 - vii. Копіюємо звіт в меню звітів (**Реєстр / Копіювати F5**).
 - viii. Звіт готовий до виконання.

1.4.4. Засоби створення вихідних форм

Комплекс дозволяє самостійно створювати вихідні форми, використовуючи як вбудований інструментарій підготовки вихідних форм, так і стандартні засоби настройки і генерації звітів. При цьому для розробки нової вихідної форми за основу можна взяти вже існуючу форму.

Комплекс містить дизайнер звітів - Генератор **FastReport**. Заснований на **SQL**-запитах, які відбирає дані з таблиць **Комплексу**. Перелік і наповнення таблиць представлений в додатках. При розробці вихідних форм за допомогою такого генератора потрібен досвід написання **SQL**-запитів і інформація про базу даних **Комплексу**.

Крім перерахованих коштів можуть бути використані продукти сторонніх розробників (наприклад, **Crystal Reports**), що перебувають в обігу до бази даних **Комплексу** через **SQL**-запити. Також формування звітів може бути реалізовано за допомогою об'єктів доступу до даних **Комплексу** (**LAO**) і продуктів **Microsoft Office** (**Word**, **Excel**).

Активний список шаблонів звітів налаштовується, використовуючи редактор меню. Виклик редактора меню звітів проводиться з вікна **Вибір вихідної форми** або **Реєстр вихідних форм** по комбінації клавіш **Ctrl + F4** або командою меню **Реєстр / Меню...**

Реєстр **Налаштування меню звітів** містить дві панелі: **Всі звіти** та **Обрані звіти**, які розташовуються вертикально або горизонтально (уявлення налаштовується по пункту меню **Вид / Горизонтальний / Вертикальний**). На панелі **Всі звіти** представлений повний список звітних форм, доступних в поточному модулі - поставляються розробником і налаштованих користувачем. Панель **Вибрані звіти** містить меню звітів - активний список звітних форм, що відображається у вікні **Вибір вихідної форми**. Копіювання необхідних звітів в активному списку проводиться по клавіші **F5**. Шаблони, перенесені на панель **Обрані звіти** (тобто включені в меню звітів), перейменовуються (клавіша **F4**), переміщаються в списку вгору або вниз (клавішами **Ctrl + / Ctrl + -** упорядковано за ознаками, формуючи групи звітів. Для цього вводяться заголовки груп (по клавіші **Ins**), а потім необхідні звіти копіюються в ці групи (по клавіші **F5** зі списку панелі **Всі звіти**).

Звіти, створені користувачами, видаляються по клавіші **F8**. При видаленні звіту з панелі **Всі звіти** він автоматично видаляється з **Вибраних звітів**. Звіти, що поставляються в **Комплексі** (в меню виділені чорним кольором) видалити не можна.

Після додавання файлу з шаблоном в каталог **REP** (знаходиться на серверної частини) необхідно перебудувати список звітів по комбінації клавіш **Alt + S** або через пункт меню **Реєстр / Перебудувати список звітів**, після чого шаблон з'являється на панелі **Всі звіти**.

У налаштуванні програм загальносистемні функції розділені по групах для зручності пошуку.

1.4.4.1. Виклик меню звітів

Для генерації звіту або друкованої форми первинного документа необхідно викликати меню звітних форм, доступних в даному модулі. Меню звітів викликається наступними способами:

- З будь-якого первинного документа по клавіші **F9** або за пунктом меню **Документ / Друк** викликається вікно **Вибір вихідної форми**, в якому запропоновано форми для друку поточного документа. Або з реєстру документів по комбінації клавіша **Ctrl + F9** або командою меню **Реєстр / Друк документа**.
- З будь-якого реєстру по клавіші **F9** або командою меню **Реєстр / Друк** викликається вікно **Вибір вихідної форми**, в якому запропоновано форми для друку поточного реєстру.
- За комбінації клавіш **Alt + F9** або командою меню **Звіт / Формування звіту** (якщо в даному реєстрі такий пункт є) викликається вікно **Вибір вихідної форми**, в якому запропоновано форми для друку різних звітів за даними поточної системи.
- Практично в кожній системі пропонується модуль **Звіти**, при завантаженні якого відкривається вікно **Реєстр вихідних форм**.

Перехід з будь-якого вікна **Вибір вихідної форми** до вікна **Реєстр вихідних форм** проводиться по клавіші **F3**.

Реєстр вихідних форм складається з двох горизонтальних панелей. На верхній панелі знаходиться активний список доступних шаблонів. На нижній панелі відображається коментар до звіту. Для шаблонів, створених користувачем (в меню виділені синім кольором), по клавіші **F4** проводиться редагування.

1.4.4.2. Імпорт-експорт шаблонів

Імпорт/експорт шаблонів звітів може використовуватися, наприклад, в разі, коли шаблон створений програмістом необхідно передати в робочий **Комплекс**.

Імпорт шаблонів здійснюється у вікні **Налаштування меню звітів** при активній панелі **Всі звіти** по пункту меню **Реєстр / Імпорт** або по комбінації клавіш **Alt + I**. У вікні **Імпорт шаблонів** вказується ім'я каталогу, в якому знаходиться імпортований звіт. У списку файлів з шаблонами по клавіші **Пробіл** виділяється необхідний.

Експорт шаблону здійснюється по комбінації клавіш **Alt + E**. У вікні **Експорт шаблону** вказується шлях до каталогу, в який буде поміщений файл шаблону.

1.4.4.3. Генерація звітів

Для генерації звіту або друкованої форми документа викликається вікно **Вибір вихідної форми** або **Реєстр вихідних форм**, вибирається найменування необхідного звіту і натисканням клавіші **Enter** підтверджується вибір.

Якщо відобразиться діалогове вікно для уточнення додаткових параметрів звіту (наприклад, період або валюта звіту) заповнюються або вибираються з довідників необхідні дані. Параметри звіту можуть бути обов'язковими або необов'язковими. До тих пір, поки не заповнені обов'язкові параметри, кнопка **ОК** залишається неактивною. Необов'язкові параметри призначені для різних вибірок (наприклад, вибір по контрагенту). Якщо параметр не заповнений, звіт виводиться для всіх значень даного параметра (в даному випадку - по всіх контрагентах).

Після обробки даних з'являється вікно **Генератора звітів** з готовим звітом - текстовим або графічним, в залежності від виду обраного шаблону.

1.4.4.3.1. Перегляд і друк звітів.

Генератор звітів відкривається спільно з вікном **Комплексу**. Тому, сформувавши звіт, його переглядають, і, при необхідності, є можливість переключитися в вікно **Комплексу**, запустити формування іншого звіту і знову повернутися в графічний генератор звітів. Генератор підтримує багатовіконність, тому можливий перегляд в ньому одночасно декількох готових звітів.

Графічний **Генератор звітів** не просто забезпечує власне генерацію звітів за готовими шаблонами з використанням призначених для користувача даних. З багатофункціонального інтерфейсу **Генератора звітів** доступні операції збереження сформованого звіту, відкриття будь-якого звіту з архіву звітів для перегляду, експорту звіту в формати **Word** і **Excel** документів, друку всього звіту, або окремих його частин.

Якщо осередок був визначений як редагований, то на екрані його вміст виділяється синім кольором.

Інтерфейс графічного **Генератора звітів** дозволяє редагувати текстову і графічну інформацію. Для цієї мети в звіт вбудована панель, що забезпечує основні функції редагування і викликається по кнопці **Виклик редактора** панелі **Вид**.

Існує можливість змінити шрифт і накреслення символів, орієнтир для вирівнювання рядків або чисел, обрамлення навіть для тих осередків звіту, які не були зазначені як редаговані при налаштуванні шаблону звіту. При цьому доступний весь набір **Windows** - шрифтів, встановлених на комп'ютері. При необхідності повністю видаляються окремі фрагменти звіту. Виділивши в осередку окремі символи, копіюють в буфер обміну, вирізають або видаляють їх, скориставшись контекстним меню, що відкривається натисканням правої кнопки миші. Символи з буфера обміну вставляються в інший фрагмент звіту, за допомогою того ж контекстного меню. Необхідні ділянки тексту звіту знаходять з використанням функції **Пошук тексту**, доступної через пункт меню **Вікно** або по клавіші **F7**.

Для друку звіту використовується панель **Друк**. На відміну від друку текстових звітів, при друку графічних звітів спеціальні настройки принтерів в **Комплексі** не мають значення, так як використовуються стандартні настройки **Windows**.

Якщо осередки шаблону виходять за робочу область листа, то при друку генератор видає попередження про те, що звіт буде надрукований в повному обсязі.

Встановлення шрифту для друку штрих-кодів в звітах.

Для встановлення шрифту досить перевстановити робочу станцію (після встановлення пакету) або вручну встановити в **Комплексі** шрифт EanGnivc.ttf. Шрифт знаходиться в папці WIN пакета. Встановлення шрифту в Windows: Пуск (Start) / Налаштування (Settings) / Панель управління (Control Panel) / Шрифти (Fonts) / Файл (File) / Встановити шрифт (Install New Font).

1.4.4.4. Експорт в звітах

У генераторі **звітів FastReport** реалізована можливість експорту готового звіту в **Microsoft Excel** або **Microsoft Word**. Для цього призначена панель **Експорт**. При експорті в **Microsoft Word** текст поміщається в таблицю, рядки та клітинки якої відповідають рядкам і осередкам в звіті. Файл зберігається у форматі **RTF**. При експорті в **Excel** всі осередки експортуються в загальному форматі.

Всі звіти, при необхідності, експортуються в **M.E.Doc** (функціями передачі даних для експорту).

1.4.4.5. Архів звітів

Якщо в налаштуванні шаблону звіту був встановлена ознака **Розміщувати в архів** або **Видавати запит**, то при поверненні з режиму перегляду звіту в попередній режим, сформований звіт буде збережений в архіві звітів відповідно автоматично або з попереднім запитом.

Звіти, поміщені в архів звітів, проглядаються і виводяться на друк таким же чином, як і при їх генерації.

Доступ до архіву звітів здійснюється з пункту меню **Сервіс / Архів звітів** або по комбінації клавіш **Shift + F9** в будь-якому модулі. Програма виводить список всіх сформованих і збережених в архіві звітів із зазначенням найменування звіту, часу його формування та довжини звіту в листах. По клавіші **Enter** викликається режим перегляду і друку звіту.

Архів звітів забезпечений функцією сортування по найменуванню, дату і час. Для включення функції сортування з пункту меню **Вид** архіву звітів вибирається пункт **Сортування** і спосіб

сортування (**Код, Найменування** або **Дата, час**). Список звітів в архіві сортується за вказаною параметру. Непотрібні звіти видаляються по клавіші **F8**.

1.4.4.6. Шаблони

Разом з **Комплексом ISpro** розробниками поставляються шаблони основних звітів і друкованих форм. Набір цих шаблонів, як правило, достатній для початку роботи. Однак для обліку специфіки конкретного підприємства, необхідно налаштовувати готові і створювати нові шаблони. Новий шаблон створюється через **Налаштування меню** і на активній панелі **Всі звіти** по клавіші **Ins** (або по пункт меню **Реєстр / Створити**). У вікні **Створення форми звіту** обирається один з можливих **Типів** (**Експорт, FastReport -Звіт, XML - звіт, Excel - звіт, Word -Звіт, Звіт OLAPі звіти Excel 2007,2010**)

Звіт - вибирається вид створення звіту - **Звіт Fast Report**

З запропонованого списку (викликається по клавіші **F3**) обирається **Шаблон**. Список можливих шаблонів залежить від системи і модуля, де проводиться налаштування звіту. Важливо зробити правильний вибір, оскільки саме обраний шаблон визначає спосіб складання звіту, список доступних даних і набір вбудованих запитів.

За кнопці **ОК**, в залежності від обраного **Типу**, запускається текстовий або графічний дизайнер звітів або ж спеціальний режим настройки експорту даних.

За комбінацією клавіш **Shift + Ins** при використанні елементів готового шаблону, курсор встановлюється на звіт, який слід взяти за основу (або вибирається пункт меню **Реєстр / Породити...**). Вибирається **Тип** звіту, який необхідно породити за зразком вибраного шаблону, вказується його **Найменування**. Всі елементи шаблону-зразка копіюються в новий шаблон. Далі в ньому проводяться необхідні зміни.

Виклик **Дизайнера звітів** для редагування готового шаблону здійснюється з панелі **Всі звіти** в **Настройці меню звітів** по клавіші **F4** або з вікна **Вибір вихідної форми** по клавіші **F4**. Редагуються (і видаляються) тільки шаблони звітів, налаштовані користувачем. У списку шаблонів вони виділені синім кольором. Шаблони, що поставляються розробником (в списку відзначені чорним кольором), не коригуються і не видаляються.

Для того щоб навчитися проектувати шаблони, необхідно представляти структуру шаблону і процес формування звіту з шаблону.

Завдання звіту - вивести сукупність даних, відповідних даної задачі, в наочній, зручною для аналізу формі.

1.4.4.6.1. Фрагменти звіту

Шаблон звіту складається з окремих фрагментів, в які можуть бути включені поля бази даних. Кожен з фрагментів виконує певне завдання в процесі генерації звіту і представлений в звіті певним числом рядків. Фрагменти розділяються на типи: **Шапка** і **Підніжжя** друкованого аркуша, **Заголовки**, **Підсумки** і **Детальні**.

При проектуванні шаблону фрагменти можуть бути об'єднані в **Групи** з рівнями підпорядкування, які утворюють ієрархічну структуру. У кожену групу безпосередньо включається довільне число фрагментів, крім детальних. При цьому в групі зобов'язаний бути присутнім хоча б один детальний фрагмент або підпорядкована група.

1. Детальний рядок

Фрагмент, який відповідає певній групі записів бази даних, визначених для даного виду шаблону розробником. Кожному шаблону відповідають свої детальні рядки, які включають в звіт. Їх кількість обмежена, вони мають свої назви, які визначаються шаблоном. Кожному детальному рядку відповідають свої поля, які будуть виводитися тільки в цьому рядку.

Фрагмент детальної інформації в залежності від найменування виводиться стільки разів, скільки записів зберігається в БД.

Розділ бази даних, відповідний призначенням звіту визначається розробником.

2. Тема

Фрагмент для виведення заголовка таблиці і підзаголовків груп даних. Фрагмент заголовка видається при зміні значень полів, визначених як ключі групи, в яку вони входять.

Фрагмент **Тема** виводиться на початку своєї групи перед **Детальними** фрагментами.

3. Шапка

Фрагмент **Шапка** будь-якої групи завжди виводиться на початку листа і призначений для оформлення верхніх колонтитулів. Якщо видача фрагмента відбувається в середині або кінці листа, фрагмент буде надрукований на новій сторінці. Йому можуть передувати тільки фрагменти **Шапка** охоплюють груп.

4. Підсумок

Фрагмент, який виводиться після кожної групи даних або в кінці всього звіту, використовується для формування підсумкових значень. Видається при зміні значень полів, визначених як ключі групи, в яку він входить.

5. Підніжжя

Фрагмент **Підніжжя** виводиться в кінці листа, призначений для оформлення нижнього колонтитула або кінця сторінки. Після нього можуть слідувати тільки фрагменти **Підніжжя** охоплюють груп.

1.4.4.6.2. Угруповання даних

Дані, що виводяться в детальний фрагмент звіту, групуються за певною ознакою (**ключу**) таким чином, що кожна група містить всі записи при певному значенні **ключа**. Наприклад, угруповання ТМЦ по складах, так, щоб в заголовку кожної групи стояла назва складу, в групі виводилися записи, відповідні поточному складу, а в підсумковому рядку - підсумкові дані по складу. Механізм угруповання реалізований наступним чином. В процесі проектування шаблону, при створенні групи, для неї визначається так званий ключ закриття групи - сукупність найменувань полів, за якими повинна проводитися угруповання (в нашому прикладі - код складу). У процесі виведення **Детального** фрагмента проводиться перевірка значення ключа. Якщо при виведенні чергового запису значення ключа змінилося, то це призводить до завершення групи і висновку до звіту підсумку по групі. Якщо вказано кілька ключів групи, то, перш за все, дані розбиваються по першому значенню ключа, потім всередині цього розбиття відбувається угруповання по другому ключу і так далі.

1.4.4.6.3. Табульовані осередки та сортування даних

Сукупність даних, що виводиться в детальний фрагмент, може мати змінну структуру. Наприклад, кожної товарно-матеріальних цінностей на складі можуть бути присвоєні додаткові характеристики (країна-виробник, матеріал, марка, сорт і так далі), причому для різних груп ТМЦ кількість і значення цих характеристик можуть бути різними. Таким чином, з'являється необхідність формування в таблиці даних звіту додаткових (табульованих) колонок з найменуваннями і значеннями, залежними від групи даних, що виводяться. Це реалізується в такий спосіб: в шаблоні звіту для груп, в яких міститься детальний фрагмент з полями змінної структури, визначається так званий ключ табуляції - найменування цих **полів**. Ключ табуляції поширює свою дію на всі групи підлеглого рівня. **Генератор звітів** виводить поля ключа табуляції в послідовні, зміщені вздовж рядка, значення, так що в результаті утворюються необхідну кількість додаткових колонок.

Користувач має можливість також задати сортування даних за певним параметру (Наприклад, здійснити сортування за кількістю або за ціною). Це здійснюється зазначенням ключа сортування при проектуванні шаблону.

Подробиці створення табульованих осередків в звіті див. у описі дизайнера.

1.4.4.6.4. Програми користувача

Для створення простого звіту досить налаштувати структури звіту, ввести його зовнішній вигляд і визначити поля виведення даних. Якщо в формі звіту застосовуються реквізити користувача або потрібна більш складна настройка, наприклад, завдання власних запитів, фільтрів, сортування, необхідні алгоритми налаштовуються за допомогою програм користувача.

Три види програм мають фіксовані імена: **Ініціалізація** (Програма ініціалізації звіту), **Обробка сортування користувача**, (Програма фільтрації і сортування звіту) і **Завершення** (Програма завершення звіту). Ці програми не вимагають підключення до фрагментів звіту.

Призначені для користувача процедури, задані в програмі під ім'ям **Ініціалізація**, викликаються перед початком формування звіту. Програма може використовуватися для установки початкових значень полів або опису функцій для виведення додаткових запитів перед формуванням звіту.

Процедури програми **Завершення** виконуються на стадії завершення звіту. Вони можуть бути використані, наприклад, для виведення будь-яких діагностичних повідомлень за результатами формування звіту.

У програмі з найменуванням **Обробка сортування користувача** задаються умови додаткової фільтрації даних, вона викликається при формуванні кожного запису тимчасової бази, якщо застосовується сортування користувача.

Всі інші процедури, зокрема, алгоритми розрахунку реквізитів користувача, накопичення підсумків і т.п. задаються в призначеній для користувача програмі з довільним найменуванням. Кількість призначених для користувача програм, заданих в одному шаблоні, не обмежена.

4 Для того щоб алгоритм користувацької програми виконувався при генерації звіту, вона обов'язково повинна бути підключена до фрагментів типу **Детальна рядок**, **Підсумок** або **Підніжжя**.

При написанні програм використовується спеціальна мова типу **Basic** (описана в п.2.3).

У програмі **Обробка сортування користувача** для фільтрації даних використовується спеціальний системний реквізит **REC_FILTER - Ідентифікатор відфільтрованої записи**. За замовчуванням значення реквізиту дорівнює 0, що означає включення запису в звіт. Для того щоб відфільтрувати запис, йому необхідно присвоїти значення 1. Насправді той же результат отримують, якщо в призначеній для користувача програмі використовувати реквізит **_ RepSupress / Ознака невидачі фрагмента на друк**. Однак, виділення умов відбору записів по можливості в окрему програму Обробка сортування користувача дозволяє підвищити наочність шаблону і полегшує його

настроювання. Для програмного придушення виведення підсумків на друк необхідно в програмі користувача, прив'язаною до підсумкового фрагменту,

встановити **_RepSupress = 1**.

Приклад.

Для того, щоб в звіті по картотеці не виводити картки з нульовою кількістю, слід заповнити програму Обробка сортування користувача:

```
IF SPKrtQt = 0 THEN REC_FILTER = 1 ELSE REC_FILTER = 0
```

Якщо в даному шаблоні не планується введення власної нестандартної сортування, введіть програму користувача:

```
IF SPKrtQt = 0 THEN _RepSupress = 1 ELSE _RepSupress = 0
```

УВАГА! Не забувайте виконати прив'язку програми користувача до детального рядку!

Підключення призначених для користувача програм до фрагментів звіту

Призначені для користувача програми можуть підключатися до фрагментів типу **Детальна рядок**, **Підсумок** або **Підніжжя**.

Доступ до полів фрагментів останніх двох типів має деякі особливості і здійснюється через так звані аліасні поля.

Аліасні поля призначені для виведення у підсумкових фрагментах таких полів, значення яких розраховуються програмою користувача. Причому програма користувача прив'язана до підсумкового фрагменту або підніжжя і викликається під час їх виведення. Аліасні поля є, по суті, псевдонімами, які підставляються у формули замість полів підсумкового фрагмента або підніжжя. Ім'я створюваного аліасного поля формується з імені вихідного поля шляхом додаванням префікса " RA _". Ці поля слід використовувати в програмах для роботи з підсумками або підніжжям.

1.4.5. Дизайнер FastReport .NET Core

FastReport.Net – генератор звітів з широкими можливостями.

FastReport.Net являє собою веб-сервіс з набором веб-методів. Це можуть бути чотири базові функції CRUD, які підтримуються протоколом HTTP. Вони реалізуються запитами: GET – читання, POST – створення, PUT – зміна, DELETE – видалення.

1.4.6. Використання каталогу USER_DATA

У звітах **FastReport**, написаних на мові Pascal, реалізована можливість використання директиви Uses, яка дозволяє підключення до програми звіту зовнішніх модулів, що представляють собою текстові файли з процедурами і функціями на мові Pascal. Це дозволяє оформляти часто використовувані процедури і функції в окремі бібліотеки.

Пошук підключаються файлів за замовчуванням буде проводитися в підкаталозі UNITS, каталогу USER _ DATA серверної частини **Комплексу**.

Каталог USER _ DATA з усім вмістом включається в резервну копію.

Приклад модуля myUnit. pas:

```
procedure s_Hello;  
  
begin  
  
ShowMessage ('Hello from MyUnit.pas');  
  
end ;  
  
begin  
  
end.
```

Здійснюється виконання скриптів. Файли для користувача скриптів зберігаються в каталозі USER_DATA \ SQL, серверної частини **Комплексу**. Файли мають розширення SQL. Призначені для користувача скрипти виконуються при відновленні резервної копії і при оновленні серверних об'єктів в БД підприємства.

Список скриптів в файлі USER_DATA \ SQL \ USERSCRIPTS.LST серверної частини **Комплексу**.

Приклад USERSCRIPTS.LST:

```
Script1.Sql  
  
Script2.Sql  
  
Script3.Sql  
  
Script4.Sql
```

Порядок виконання скриптів визначається порядком проходження в файлі.

Приклад користувальницького скрипту для MSSQL Script1.Sql:

Create table TESTTABLE

(ID int identity (1,1),

Operation varchar (50))

GO

Примітка.

Для СУБД MSSQL обов'язково в кінці скрипта написати "GO".

Для СУБД Oracle обов'язково в кінці скрипта написати "/".

1.4.7. Експорт звітності

Для експорту звітів, побудованих за допомогою дизайнера необхідно скласти програми, які запускаються при формуванні фрагментів звіту.

Програми для звітів складаються на мові PascalScript. У програмах доступні функції, необхідні для формування файлу електронного документа. Повний перелік функцій наведено нижче.

Загальні правила побудови програми:

- На початку, при ініціалізації експорту, викликається функція MNSOpen, для ініціалізації файлу електронного документа.
- Потім формується заголовок файлу.
- У процесі видачі детальних рядків має бути сформовано вміст форми документа.
- В кінці експорту потрібно сформувати кінець фрагмента форми, кінець файлу і викликати функцію MNSClose, яка завершить сеанс роботи з файлом електронного документа.
- Програми формування файлу експорту прив'язуються до детальним фрагментами звіту. Можуть бути використані будь-які фрагменти (заголовки, детальні, підсумки).

Перелік доступних констант, функцій і процедур

Константи

Константи режимів відкриття файлу

mnsomAppend: Byte = \$ 01;

Дописати в кінець файлу, якщо існує.

mnsomCreate: Byte = \$ 00;

Перезаписати файл, якщо існує.

mnsomDialog: Byte = \$ 10;

Видати діалог вибору файлу.

Константи роздільників груп

mnsdlmBlock: String = '###';

Кінець блоку.

mnsdlmFile: String = '===';

Ознака кінця файлу.

mnsdlmPart: String = '@@@';

Кінець фрагмента.

Функції та процедури

1.4.7.1.

DOCGetStr (Nam: String): String;

DOCGetStr (" ");

Отримати строкове значення поля з документа.

MNSClose ;

Закрити поточний відкритий файл.

MNSGetFL_ID (INN: String): String;

MNSGetFL _ ID (" ");

INN - ІПН фізичної особи

Дати ідентифікатор ФО.

MNSGetFormDetailNm (RZD, STR, GRF: String): String ;

MNSGetFormDetailNm (" ", " ", " ");

RZD - п'ятизначний номер (код) розділу форми

STR - п'ятизначний номер (код) рядку форми

GRF - двозначний номер граfi форми

Дати найменування показника форми.

MNSGetMNS _ ID (MNS: String): String ;

MNSGetMNS _ ID (" ");

MNS - коду інспекції за класифікатором СОНО

Дати ідентифікатор МНС.

MNSGetUL_ID (INN: String; KPP: String): String;

MNSGetUL_ID (" ", " ");

INN - ІПН організації

KPP - код причини постановки на облік, якщо " - не виводиться

Дати ідентифікатор ЮЛ.

MNSOpen: Boolean;

MNSOpen;

Відкриває останній відкритий файл, в разі помилки згенерує виняток.

MNSOpenFile (Filename: String; Mode: Integer): Boolean;

MNSOpenFile (FileName, Mode);

FileName - ім'я файлу

Mode - режим відкриття

mnsomCreate - перезаписати файл, якщо існує

mnsomAppend - дописати в кінець файлу, якщо існує

mnsomDialog - видати діалог вибору файлу

mnsomWin1251 - кодування Windows

Відкриває файл, в разі помилки згенерує виняток.

MNSPutCommonInfo (DocNum: String; DocDate: TDateTime; DocPredst: String);

MNSPutCommonInfo (" ");

DocNum - номер документа

DocDate - дата формування документа

DocPredst - параметр "документ видається"

Вивести в поточний відкритий файл загальні відомості інформаційної частини повідомлення.

MNSPutDate (AParam: String; AValue: TDateTime);

MNSPutDate (" ");

Вивести поєднання параметр-значення в поточний відкритий файл. В кінець рядка дописується пара символів \$ 0 d \$ 0 a. Рядок передається з WIN кодування в OEM кодування.

MNSPutDelimiter (ADelimiter: String);

MNSPutDelimiter (" ");

Вивести роздільник в поточний відкритий файл.

MNSPutDetail (RZD, STR, GRF, Val: String);

MNSPutDetail (" ", " ", " ", " ");

RZD - п'ятизначний номер (код) розділу форми

STR - п'ятизначний номер (код) рядку форми

GRF - двозначний номер граfi форми

Val - значення показника

Вивести показник форми.

MNSPUTDOCID (A_ID: String; DocNum: String; ADate: TDateTime = 0);

MNSPutDocID (" ", " ");

A _ ID - ідентифікатор ЮЛ, ФЛ або МНС

DocNum - номер документа

ADate - дата, за замовчуванням - поточна

Вивести ідентифікатор документа в поточний відкритий файл.

MNSPUTFILEID (A_ID: String; ADate: TDateTime = 0);

MNSPutFileID (" ");

A_ID - ідентифікатор ЮЛ, ФЛ або МНС '+ # 13 +

ADate - дата, за замовчуванням - поточна

Вивести ідентифікатор файлу в поточний відкритий файл.

MNSPUTNPFLINFO ;

Вивести в поточний відкритий файл відомості про платника податків - юридичну особу.

MNSPUTNUMBER (AParam: String; AValue: Extended; ALen: Integer; AMT: Integer; AFill0: boolean);

MNSPutNumber (" ", „);

ALen - довжина всього числа з крапкою і знаком -

AMT - к-ть знаків після зайнятої

AFill 0 - доповнювати число до довжини нулями зліва

Вивести поєднання Параметр-Значення в поточний відкритий файл. В кінець рядка дописується пара символів \$ 0D \$ 0A. Рядок перекодується з Win-кодування в Oem.

MNSPUTOVERHEAD (KolDoc: Word ; MNSVer: String);

MNSPutOverhead ();

KolDoc - кількість документів

Вивести в поточний відкритий файл службову частину повідомлення.

MNSPUTPARAM (AParam: String; AValue: String);

MNSPutParam (" ");

Вивести поєднання Параметр-Значення в поточний відкритий файл. В кінець рядка дописується пара символів \$ 0D \$ 0A. Рядок перекодується з Win кодування в Oem.

MNSPUTSTRING (AParam: String; AValue: String; ALen: Integer);

MNSPutString (" ");

ALen - довжина всього числа з крапкою і знаком -

Вивести поєднання Параметр-Значення в поточний відкритий файл значення - строкового типу. Значення форматується відповідно до довжини, якщо потрібно - буде скорочуватися. В кінець рядка дописується пара символів \$ 0D \$ 0A. Рядок перекодується з Win кодування в Oem.

MNSPUTTEXT (AText: String);

MNSPutText (" ");

Вивести рядок в поточний відкритий файл. В кінець рядка дописується пара символів \$ 0D \$ 0A. Рядок перекодується з Win кодування в Oem.

SETKOLDOK (Kol: Integer);

SetKolDok ();

Встановити кількість документів.

SETKOLPOK (Kol: Integer);

SetKolPok ();

Встановити кількість показників форми.

STRINGTOCOMMA (AText: String): String;

StringToComma (" ");

Замінити прогалини на коми.

Приклад використання функцій експорту зовнішньої звітності в звіті Fast Report

1.4.7.2.

Відкрити на редагування будь-який звіт. У обробнику події OnStartReport робимо відкриття файлу, наприклад:

MNSOpenFile ('C: \ MNStempoFile', mnsomCreate);

У обробнику події OnStopReport робимо закриття файлу:

MNSClose ();

У місцях формування даних для звіту використовуємо функції вивантаження, наприклад:

MNSPutText (Memo4.Text);

MNSPutDelimiter (mnsdImPart);

Таким чином, результируючий файл може виглядати так:

var curs: real;

procedure Memo6OnBeforePrint (Sender: TfrxComponent);

begin

if <main. "VLCUR_KOT"> = <cross. "CURTYPE_RCD">

then curs: = <main. "VLCUR_CURS">

else curs: = 0;

end;

```
procedure OnStartReport (Sender: TfrxComponent);  
  
begin  
  
  MNSOpenFile ('C: \ MNStempoFile', mnsomCreate);  
  
  MNSPutText (' Тема звіту ');  
  
  MNSPutDelimiter (mnsdImPart);  
  
end;  
  
procedure OnStopReport (Sender: TfrxComponent);  
  
begin  
  
  MNSPutDelimiter (mnsdImPart);  
  
  MNSPutDelimiter (mnsdImBlock);  
  
  MNSPutDelimiter (mnsdImFile);  
  
  MNSPutText (' Фінал звіту ');  
  
  MNSClose ();  
  
end;  
  
procedure Memo3OnAfterData (Sender: TfrxComponent);  
  
begin  
  
  MNSPutText (Memo3.Text);  
  
end;  
  
procedure Memo4OnAfterData (Sender: TfrxComponent);  
  
begin  
  
  MNSPutText (Memo4.Text);  
  
  MNSPutDelimiter (mnsdImPart);  
  
end;
```

```
procedure Memo14OnAfterData (Sender: TfrxComponent);
```

```
begin
```

```
MNSPutText (floatToStr (curs) + " + Memo14.Text);
```

```
end;
```

```
begin
```

```
end.
```

В результаті виконання звіту формується наступний файл - C: \ MNStempoFile.

Лістинг файлу:

заголовок звіту

@@@

USD

Долар США

Множник курсу: 1

@@@

5.05 5.05000

5.05 5.05000

...

5.05 5.05000

EUR

ЄВРО

Множник курсу: 1

@@@

6.4321 6.43210

6.3933 6.39330

...

6.405925 6.40592

RUR

Російський рубль

Множник курсу:

@@@

0.18857 0.18857

0.18847 0.18847

...

0.1889 0.18890

@@@

###

===

фінал звіту

1.4.8. Імпорт файлів MS Office Word (*.doc) в FastReport

Для прискорення реалізації звітів використовується імпорт файлів **MS Office Word (*.doc)** в **FastReport**. У редакторі звітів на панелі інструментів **Мастеру** за допомогою кнопки **Імпорт з MS Office Word** відкривається вікно імпорту.

Імпорт з MS Office Word проводиться згідно налаштування:

Тип - Повний, Тільки таблиці, В обрану таблицю, Тільки текст;

1. **Сторінками - Все** або **обрана** (в першому випадку відсутні сторінки в FR створюються автоматично. У другому - необхідну сторінку в FR можна створити прямо в вікні імпорту);
2. **Текст - по абзацах** або **за словами**.

У MS Word немає поняття «рядок» в звичному «досовському» трактуванні, а є тільки поняття «абзац», який є одним рядком, який переносить автоматично по деяким законам, наприклад, в залежності від ширини листа. При цьому в **MS Word** дозволяється форматування (зміна шрифту, стилю і т.д.) будь-якій частині тексту, аж до одного символу. У свою чергу в **FastReport** форматування допустимо тільки в межах **TfrxMemoView**. Тому реалізовано два типи імпорту тексту:

- **По абзацах** (абзац Word = TfrxMemoView);
- **П про словами** (слово Word = TfrxMemoView).

Перший тип працює швидко, але форматування тексту може не відповідати оригіналу в разі, якщо в межах абзацу застосовуються різні формати тексту.

Другий тип працює значно повільніше, але форматування відповідає аж до окремих слів.

Після імпорту може знадобитися незначна коректування звіту.

1.4.9. Розробка звітів MS Office (Excel, Word)

Комплекс підтримує створення, редагування і запуск шаблону **Excel**. Для правильної взаємодії з **Комплексом** при розробці шаблону слід дотримуватися певних правил:

- на зовнішній вигляд і вміст шаблону не потрібно встановлювати обмеження;
- VBA програми формування звіту повинні починати роботу всередині підпрограми ReportMain;
- підпрограма ReportMain служить «точкою входу» в програмний блок звіту. **Комплекс** завантажує шаблон, відшукує в його модулях підпрограму ReportMain, запускає її і чекає завершення підпрограми. Параметром підпрограми ReportMain передається менеджер COM об'єктів **Комплексу**. За допомогою менеджера можливий доступ до COM об'єктів **Комплексу** для отримання інформації, необхідної звіту. Об'єкт менеджера можна використовувати до закінчення виконання підпрограми ReportMain, після її закінчення об'єкт буде зруйнований і всі звернення до нього приведуть до помилки;
- вся програма формування звіту повинна розташовуватися всередині підпрограми ReportMain.

Створення даних звітів починається з вибору при створенні форми звіту типу - Excel звіту або Word - звіту.

Реалізовано новий тип звітів **Excel 2007, 2010** для реалізації додаткового функціоналу, який надає **Microsoft Excel 2007 та 2010**. Ці звіти мають розширення **xltm** (системні звіти) і **xlum** (призначені для користувача звіти). Даний тип звітів працює тільки в середовищі **Microsoft Excel 2007** і вище.

У рядку **Шаблон** відображаються таблиці, склад яких залежить від системи в якій створюється звіт.

У рядку **Найменування** - вводиться найменування створюваного звіту.

Після натискання клавіші **Ок** проводиться запуск програми (Excel або Word відповідно). Відкривається файл що є шаблоном звіту. Для формування звіту необхідно по пункту меню програми **Сервіс / Макрос / Редактор Visual Basic** запустити редактор формування програми звіту.

У редакторі вже сформований кістяк програми. Далі за допомогою **Visual Basic** створюється програма побудови звіту. Доступ до даних здійснюється за допомогою SQL запитів, COM -об'єктів.

Для формування образу шаблону звіту його можна копіювати з існуючих звітів (по комбінації клавіш **Ctrl + C, Ctrl + V**).

1.4.9.1. Приклад програми Excel звіту

Приклад програми Excel звіту (VBA script)

'Звіт складається з даних двох таблиць

'Ptngprk - список груп контрагентів

'Ptnrk - список контрагентів

'Зв'язок між таблицями

'Ptnrk.ptngrpk_rcd = ptngrpk.ptngrpk_rcd

'Поля таблиць

'ptngrpk

'Ptngprk_rcd - ід записи групи

'Ptngpr_cdk - код групи

'Ptngpr_nmk - найменування групи

,

'ptnrk

'Ptngprk_rcd - ід групи

'Ptn_cd - код контрагента

'Ptn_nm - найменування контрагента

,

'Порядок побудови звіту:

'Сканується список груп контрагентів

'Для кожної групи вибирається список відповідних контрагентів

'Примірник менеджера COM об'єктів

'Необхідний для запиту примірників COM об'єктів **Комплексу**

Public ComManager As Object

'Generated code.

,

'Головна точка входу в програму побудови звіту

'Ця процедура буде викликана **Комплексом**

'Після відкриття шаблону звіту

Sub ReportMain (Mgr As Object)

On Error GoTo ErrorHandler

'Запам'ятати екземпляр об'єкта

'Переданий **Комплексом**

Set ComManager = Mgr

' Запуск побудови звіту

MakeReport

Exit Sub

ErrorHandler:

MsgBox ("Error:" & Chr (13) & Err.Description)

End Sub

Private Sub MakeReport ()

' Групи контрагентами

Dim PtnGrpObj As Object

' Контрагенти

Dim PtnObj As Object

' Лист звіту

Dim RepSheet As Worksheet

'Початкова рядок листа звіту

Dim row As Integer

'В разі виникнення помилки буде видано повідомлення з текстом помилки

On Error GoTo ErrorHandler

'Лист книги, на якому буде побудований звіт

Set RepSheet = ActiveSheet

'Запросити у менеджера екземпляри двох об'єктів виконання SQL запитів

Set PtnGrpObj = ComManager.GetObjByName ("Sys", "ISysSqlQuery")

```
Set PtnObj = ComManager.GetObjByName ("Sys", "ISysSqlQuery")
```

```
' Запит для читання груп
```

```
PtnGrpObj.Text = "select ptngrpk_rcd as rcd, ptngrp_cdk as cd, ptngrp_nmk as nm from ptngrpk"
```

```
' Запит для читання контрагентів групи
```

```
PtnObj.Text = "select ptn_cd as cd, ptn_nm as nm from ptnrk where ptngrpk_rcd =: grp_rcd"
```

```
'Відкрити запит для груп
```

```
PtnGrpObj.OpenObj
```

```
'Сканувати групи
```

```
If PtnGrpObj. First Then
```

```
row = 3
```

```
Do
```

```
'Висновок найменування групи
```

```
RepSheet. Range (" B " & CStr (row) & ": C " & CStr (row)). Merge
```

```
RepSheet.Range ("A" & CStr (row) & ": C" & CStr (row)). Font.Bold = True
```

```
RepSheet.Range ("A" & CStr (row) & ": C" & CStr (row)). BorderAround (xlContinuous)
```

```
RepSheet.Range ("A" & CStr (row)). Value = PtnGrpObj.GetFieldValue ("cd")
```

```
RepSheet.Range ("B" & CStr (row)). Value = PtnGrpObj.GetFieldValue ("nm")
```

```
row = row + 1
```

```
'Встановлення коду групи для запиту списку контрагентів
```

```
Call PtnObj.SetParamValue ("grp_rcd", PtnGrpObj.GetFieldValue ("rcd"))
```

```
'Відкрити запит списку контрагентів
```

```
PtnObj.OpenObj
```

```
' Сканувати список

If PtnObj.First Then

Do

' Висновок запису контрагента

RepSheet.Range ("A" & CStr (row) & ": B" & CStr (row)). Merge

RepSheet.Range ("A" & CStr (row) & ": C" & CStr (row)). BorderAround (xlContinuous)

RepSheet.Range ("A" & CStr (row)). Value = PtnObj.GetFieldValue ("cd")

RepSheet.Range ("C" & CStr (row)). Value = PtnObj.GetFieldValue ("nm")

row = row + 1

Loop While PtnObj.Next

End If

'Закрити об'єкт списку контрагентів

'Після закінчення читання списку для групи

PtnObj.CloseObj

Loop While PtnGrpObj.Next

End If

Exit Sub

ErrorHandler:

MsgBox ("Error:" & Chr (13) & Err.Description)

End Sub
```

1.4.9.2. Приклад програми Word звіту

Звітом є форма договору.

'Generated code.

Sub ReportMain (Mgr As Object)

Set ComManager = Mgr

Set EnvObj = ComManager.GetObjByName ("Sys", "ISysEnvironment")

'MsgBox (EnvObj.Dog_Ptn)

'Selection.Text = EnvObj.PTN_READ_INT (Selection.Text)

'Call EnvObj.PTN_READ_INT (EnvObj.Dog_Ptn)

'MsgBox (EnvObj.PtnR_NmSh)

'MsgBox (EnvObj.NUMTOSTR (EnvObj.U_tttttt))

With Selection.Find

.ClearFormatting

.Text = "& <*> &"

.Replacement.Text = ""

.Forward = True

.Wrap = wdFindContinue

.Format = False

.MatchCase = False

.MatchWholeWord = False

.MatchAllWordForms = False

.MatchSoundsLike = False

.MatchWildcards = True

If.Execute Then

Do

'MsgBox (Selection.Text)

Selection.Text = EnvObj.GETDOGTXTVALUE (Selection.Text)

Selection.EndOf

Loop While Selection.Find.Execute

End If

End With

End Sub

Перелік полів для формування договору:

Поля для функції ДОГОВОРУ

Номер договору - & DOGNMR &

Вхідний номер договору - & DOGVXDNMR &

Дата договору - & DOGDATE &

Об'єкт угоди - & DOGOBJ &

Коментар - & DOGCMT &

Центр відповідальності договору - & DOGCO &

Дата початку договору - & DOGDATEBEG &

Дата кінця договору - & DOGDATEEND &

Спосіб оплати - & DOGSPOPL &

Дата оплати - & DOGDATEOPL &

Котирування валюти - & DOGVALKOT &

Валюта договору - & DOGVAL &

Курс - & DOGVALKURS &

Реквізити свого боку

Найменування боку - & DOGSIDENM 1 &

Найменування контрагента - & DOGSIDEPTN 1 &

Адреса - & DOGSIDEADR 1 &

Контактна особа - & DOGSIDECONT 1 &

Телефон - & DOGSIDEPHN 1 &

Банк - & DOGSIDEBANK 1 &

МФО - & DOGSIDEMFO 1 &

Рахунок - & DOGSIDEACC 1 &

ІПН - & DOGSIDEINN 1 &

Реквізити іншого боку

Найменування боку - & DOGSIDENM 2 &

Найменування контрагента - & DOGSIDEPTN 2 &

Адреса - & DOGSIDEADR 2 &

Контактна особа - & DOGSIDECONT 2 &

Телефон - & DOGSIDEPHN 2 &

Банк - & DOGSIDEBANK 2 &

МФО - & DOGSIDEMFO 2 &

Рахунок - & DOGSIDEACC 2 &

ІПН - & DOGSIDEINN 2 &

Поля користувача - & ім'я &

1.4.9.3.

Приклад:

Договір N & DOGNMR &

майнового найму (класична оренда)

& DOG DATE & р м. Київ

& DOGSIDEPTN1 &, в подальшому Наймодавець, в особі & DOGSIDECONT1 &, що діє на підставі Статуту, з одного боку, і & DOGSIDEPTN2 &, в подальшому Наймач, в особі & DOGSIDECONT2 &, що діє на підставі & DOGSELSPR &, з іншого боку, керуючись взаємною угодою і чинним законодавством, уклали договір про наступне.

1. ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1. За цим договором Наймодавець зобов'язується надати Наймачеві майно у тимчасове користування, а Наймач зобов'язується оплатити його і повернути майно.

1.2. Наймодавець надає Наймачеві наступне майно:

& DOGOBJ &.

1.2.1. Наймодавець передає Наймачеві наступні документи на майно: & DOGINSTR &.

Оформлення документів, необхідних для передачі майна в найм і для нормального користування майном оформлюються & DOGINSTR &.

1.3. Наймодавець надає Наймачеві майно у тимчасове користування строком на & DOGINSTR &.

1.4. Крім вищевказаного Наймодавець надає Наймачеві наступні послуги: & DOGINSTR &.

1.5. Наймач користується отриманим за цим договором имуществом самостійно в місці: на свій розсуд.

1.6. Охорона, зберігання, обслуговування і т.п. здійснюється & DOGINSTR & & DOGINSTR &, засобами & DOGINSTR &, за рахунок & DOGINSTR & & DOGINSTR &, в наступному порядку: & DOGINSTR &.

1.7. Власником переданого в оренду майна є & DOGINSTR &.

1.8. Майно передається Наймачеві в місці: & DOGINSTR & і його здача-приймання оформляється & DOGINSTR & (Додаток N 1). (Розпискою, актом, протоколом)

1.9. Наймач після закінчення дії цього договору зобов'язується повернути майно, отримане за цим договором, наймодавця в місці: & DOGINSTR &, в стані: & DOGINSTR &, в терміни: & DOGINSTR &

& а здача-приймання майна оформляється & DOGINSTR & (Додаток N 2). (Розпискою, актом, протоколом)

1.10. Наймач зобов'язується користуватися найманим майном в порядку: & DOGINSTR &.

(Вказати мету найму, порядок користування та режим і т.п.)

2. ВЗАЄМНІ РОЗРАХУНКИ СТОРІН

2.1. За користування майном Наймач зобов'язується виплатити наймодавця найману плату в розмірі & DOGSUM & рублів за & DOGINSTR &.

2.2. Крім вищевказаних платежів Наймач виплачує Наймодавцю суму в розмірі & DOGINNUMB & рублів за & DOGINSTR & за надані Наймодавцем відповідно до п.1.4. послуги.

2.3. & DOGINSTR & за рахунок & DOGINSTR & здійснює страхування надається за цим договором майна на суму & DOGINNUMB & рублів в порядку & DOGINSTR &.

2.4. Наймач видає наймодавця в рахунок майбутніх розрахунків аванс в сумі & DOGINSTR & в порядку & DOGINSTR &.

2.5. За погодженням сторін в рахунок найманої плати (на весь період дії договору або на будь-який період) можуть передаватися послуги, права користування, майно і т.п.

2.6. Плата за користування майном вноситься незалежно від фактичного користування майном, якщо сторони додатково не визначать інше, крім випадків, коли Наймач не може користуватися майном з причин:

- хвороба користувача;
- форс-мажорні обставини;
- & DOGINSTR &.

2.7. Орендна плата за відповідний рік перераховується рівними частками щоквартально, не пізніше 20 числа першого місяця наступного кварталу.

2.8. Щорічно, після закінчення відповідного календарного року, сторони перевіряють правильність платежів по найманої плати і результати перевірки оформляють актом звірки розрахунків. За даними зазначеного акта в межах першого кварталу проводяться необхідні перерахунки.

2.9. Наймана плата підлягає перегляду у зв'язку зі зміною централізовано встановлюваних цін і тарифів, а також в інших випадках, передбачених чинним законодавством та додатковою угодою сторін.

3. Права та обов'язки сторін

3.1. Наймодавець зобов'язаний:

а) передати Наймачеві майно, що є предметом цього договору, в стані, що дозволяє його нормальну експлуатацію, відповідному його фактичній вартості, і цим договором;

б) крім вищевказаного, в тому ж порядку передати:

& DOGINSTR &;

в) забезпечити передачу Наймачеві нормативно-правової, нормативно-технічної та наукової інформації;

г) проінструктувати про порядок користування і посприяти наймодавця по користуванню найманим майном;

д) оформити документи & DOGINSTR & в терміни: & DOGINSTR &;

е) у разі надання несправного (некомплектного) майна замінити (доукомплектувати) його або виправити не пізніше доби з моменту виявлення відповідних недоліків.

Зазначений строк не входить до терміну для обчислення неустойки і штрафних санкцій;

ж) попередити Наймача про приховані властивості (недоліки в т.ч.) найманого майна, які можуть завдати шкоди Наймач або його майну;

з) провести за власний рахунок і власними засобами і силами & DOGINSTR & ремонт (& DOGINSTR & обслуговування) майна в терміни: & DOGINSTR &.

У разі, якщо наймана майно виходить з користування Наймача, зазначений термін в термін дії цього договору не включається;

і) Наймодавець не може достроково розірвати цей договір в односторонньому порядку;

к) & DOGINSTR &.

3.2. Наймач зобов'язаний:

- а) своєчасно проводити платежі, передбачені цим договором;
- б) використовувати отримане за цим договором майно відповідно до його цільового призначення;
- в) під час приймання майна перевірити його справність і комплектність;
- г) за власний рахунок утримувати майно в такому порядку:

& DOGINSTR & з урахуванням його природного зносу;

д) якщо наймана майно вибуває з ладу раніше повного амортизаційного терміну служби або & DOGINSTR &, то Наймач відшкодовує наймодавця недовнесених їм платежі за оренду майна. Залежно від фактичного стану майна та обставин вибуття з ладу, Наймодавець може не стягувати недовнесених найману плату;

е) & DOGINSTR &.

3.3. Наймодавець має право:

- а) отримати найману плату відповідно до цього договору;
- б) отримати інформацію про стан найманого майна;
- в) розірвати цей договір у разі, якщо Наймач порушує свої обов'язки в період користування найманим майном;
- г) & DOGINSTR &.

3.4. Наймач має право:

а) на здачу орендованого майна в піднайом, інше тимчасове користування іншим організаціям (в тому числі безоплатне), в разі:

- невикористання майна з причин: & DOGINSTR &;
- хвороби або & DOGINSTR & користувача майном;
- & DOGINSTR &.

Здача майна в тимчасове користування іншим організаціям не впливає на зміст фінансових відносин з Наймодавцем;

- б) на проведені їм віддільні поліпшення найманого майна;
- в) на компенсацію вартості здійснених ним невіддільних поліпшень найманого майна;

г) на зниження найманої плати в разі зниження споживчих властивостей майна з причин, не залежних від Наймача, на суму, пропорційну зміненим властивостям;

д) на відшкодування витрат на поліпшення майна, проведені за згодою (в т.ч. без згоди) Наймодавця, в складі: вартості матеріалів, робочої сили і *& DOGINSTR &*, в т.ч. на відновлення майна після його природного зносу;

е) *& DOGINSTR &*.

4. ВІДПОВІДАЛЬНІСТЬ

4.1. У разі невиконання або неналежного виконання зобов'язань винна сторона зобов'язана відшкодувати другій стороні збитки у відповідності з чинним законодавством *& DOGSELSPR &*.

4.2. За прострочення сплати найманої плати Наймач сплачує *& DOGINSTR &* відсотків від несплаченої суми в необхідний термін.

4.3. За прострочення передачі майна за цим договором (у т.ч. надання неякісного, некомплектного майна) Наймодавець сплачує *& DOGINSTR &* відсотка від вартості непереданого в термін майна за кожен день прострочення.

4.4. Неустойки, передбачені цим договором, стягуються незалежно від відшкодування збитків.

4.5. У разі псування або загибелі найманого майна Наймач зобов'язаний: *& DOGINSTR &*.

4.6. Виплата неустойок і відшкодування збитків не звільняє сторони від виконання договірних зобов'язань в натурі.

4.7. Додаткові заходи відповідальності: *& DOGINSTR & & DOGINSTR &*.

5. ДІЯ ДОГОВОРУ

5.1. Цей договір вступає в силу з *& DOGDATEBEG &* і діє *& DOGINSTR &* (для майна - не більше 10 років; будови - не більше 5 років; обладнання - не більше 1 року)

5.2. Після закінчення зазначеного в п. 5.1. терміну Наймач має право на пролонгацію договору на новий термін, який сторони визначають додатково на умовах не погіршують становище сторін та за умови, що Наймач протягом терміну дії цього договору не порушував прийняті на себе зобов'язання.

5.3. Договір може бути змінений або розірваний в наступному порядку: Зацікавлена сторона повинна направити іншій стороні пропозицію про це. Сторона, яка отримала таку пропозицію, повинна дати відповідь не пізніше 10 днів після одержання пропозиції. У разі недосягнення згоди щодо зміни, розірвання договору, а також у разі неодержання відповіді спір вирішується за заявою зацікавленої сторони Державним арбітражем.

5.4. Цей договір може бути розірваний за ініціативою однієї сторони і поза порядком, зазначеного в п.5.3. в наступних випадках:

- погіршення стану найманого майна з причин, не залежних від Наймача;

- & DOGINSTR &.

5.5. У разі досягнення угоди про переукладання цього договору і до укладення нового договору діють норми чинного законодавства.

6. Додаткові умови

6.1. Спори сторін вирішуються в & DOGINSTR & & DOGINSTR &

6.2. & DOGINSTR & & DOGINSTR &

6.3. & U DOG &

7. Інші умови

7.1. На відносини сторін, не врегульовані цим договором, поширюються норми чинного цивільного законодавства.

7.2. Цей договір складено в 2 примірниках: по одному кожній стороні.

7.3. До цього договору в якості невід'ємних його частин додаються:

- Додаток N 1 - акт здачі-приймання майна;

- Додаток N 2 - акт здачі-приймання палива, сировини, матеріалів, незавершеного виробництва, готової продукції, фонди економічного стимулювання.

7.4. Реквізити сторін:

наймодавець

наймач

Адреса: & DOGSIDEADR1 &

Адреса: & DOGSIDEADR 2 &

Телефон: & DOGSIDE PHN2 &

Телефон: & DOGSIDE PHN2 &

р / с & DOGSIDEACC 1 &

р / с & DOGSIDEACC2 &

Адреса: <u>& DOGSIDEADR1 &</u>	Адреса: <u>& DOGSIDEADR 2 &</u>
Телефон: <u>& DOGSIDE PHN2 &</u>	Телефон: <u>& DOGSIDE PHN2 &</u>
р / с <u>& DOGSIDEACC 1 &</u>	р / с <u>& DOGSIDEACC2 &</u>
<u>& DOGSIDEBANK1 &</u>	<u>& DOGSIDEBANK2 &</u>
<u>& DOGSIDEMFO1 &</u>	<u>& DOGSIDEMFO2 &</u>
(Підпис)	(Підпис)
м.п.	м.п.

1.4.10. Робота з COM об'єктами Комплексу

Для організації взаємодії зовнішнього застосування з середовищем **ISpro** в поставку **Комплексу** включений ряд COM об'єктів, службовців для створення з'єднання з **Комплексом** і отримання примірників об'єктів.

COM-об'єкти **Комплексу** (**Component Object Model** - компонентна об'єктна модель) призначені для організації зовнішнього доступу до прикладних функцій і даних **Комплексу**.

Все COM-об'єкти **Комплексу** відповідають вимогам автоматизації і можуть використовуватися з будь-яких мов, що підтримують роботу з інтерфейсами автоматизації.

COM об'єкти можуть містити набір предметних функцій, а так само, можуть повторювати функціональність набору даних і містити набір стандартних методів навігації і модифікації даних.

Для отримання примірника певного об'єкта зовнішню програму має виконати наступні кроки:

- Встановити з'єднання з **Комплексом**, вказавши при цьому бажане підприємство. Після успішної установки з'єднання буде запущена робоча станція **Комплексу**, яка працює в режимі COM сервера.
- в результаті успішного з'єднання зовнішню програму отримує екземпляр об'єкта - менеджера COM об'єктом. Менеджер відповідає за створення екземплярів всіх інших COM об'єктів **Комплексу**.
- зовнішню програму, за допомогою менеджера, створює необхідні COM об'єкти.

Звільнення об'єктів проводиться штатними засобами застосовуваного мови програмування.

Приклад програми на VBA, що виводить перші два рядки реєстру контрагентів.

' Менеджер COM об'єктів

Dim ComManagerObj As Object

'З'єднання з **Комплексом**

Dim ConnectObj As Object

'Запит для читання записів картотеки

Dim QueryObj As Object

On Error GoTo ErrorHandler

'Створення об'єкту для з'єднання з **Комплексом**

Set ConnectObj = CreateObject ("ISStBoot.SysStartup.1")

'Запуск створення з'єднання

'- станція в каталозі c: \ ispro

'- користувач adm

'- пароль master

Set ComManagerObj = ConnectObj.Connect ("c: \ ispro", "adm", "master")

'З'єднання встановлено

'Вхід в підприємство №1

Call ConnectObj.SelectFirm (1)

'Всі етапи з'єднання завершені, можна працювати з **Комплексом**

'Запросити об'єкт для виконання довільного SQL запиту

'Модуль об'єкта - Sys

'Найменування об'єкта - ISysSqlQuery

```
Set QueryObj = ComManagerObj.GetObjByName ("Sys", "ISysSqlQuery")
```

'Виконати запит читання картотеки контрагентів

```
QueryObj.Text = " select ptn _ nm from ptnrk "
```

```
QueryObj.OpenObj
```

'Прочитати перший запис

```
QueryObj.First
```

```
MsgBox (QueryObj.GetFieldValue ("ptn_nm")
```

' Прочитати другий запис

```
QueryObj.Next
```

```
MsgBox (QueryObj.GetFieldValue ("ptn_nm")
```

' Звільнити об'єкт

```
QueryObj.CloseObj
```

```
Set QueryObj = Nothing
```

ErrorHandler:

```
MsgBox ("Error:" & Chr (13) & Err.Description)
```

1.4.10.1. COM об'єкт для виконання SQL запитів

Об'єкт призначений для виконання довільних SQL запитів в схемі даних поточного підприємства і схемою загальносистемних даних. Облік розташування таблиці в загальній схемі даних або схемою підприємства, виконується автоматично.

Опис системних COM-об'єктів **Комплексу**

- Назва об'єкту: ISysSqlQuery

Модуль: Sys

Бібліотека типів: sys _ c. dll

склад об'єкта

BSTR Text

Читання / запис тексту SQL запиту.

SetParamValue (BSTR ParamName, VARIANT Val)

Установка значення параметра запиту.

ParamName - найменування параметра, без символу ":"

Val - значення параметра

VARIANT GetFieldValue (BSTR FieldName)

Повертає значення поля запиту.

FieldName - найменування поля

VARIANT_BOOL OpenObj

відкрити набір

VARIANT_BOOL HRESULT CloseObj

Закрити набір

VARIANT_BOOL First

Стати на перший запис

VARIANT_BOOL Last

Стати на останній запис

VARIANT_BOOL Prior

Стати на попередній запис

VARIANT_BOOL Next

Стати на наступний запис

У разі необхідності використання параметрів в запиті, параметри необхідно іменувати відповідно до правил іменування полів і випереджати параметр в тексті запиту символом ":", наприклад where rcd =: new_rcd порівнюється значення поля "rcd" зі значенням параметра "new_rcd".

Приклад використання об'єкта

```
Private ConnectObj As Object
```

```
Private ManagerObj As Object
```

```
Private QryObj As Object
```

```
Private Sub DoWork ()
```

```
On Error GoTo ErrorHandler
```

```
Set ConnectObj = CreateObject ("ISStBoot.SysStartup.1")
```

```
Set ManagerObj = ConnectObj.Connect ("D: \ bpw", "pva", "")
```

```
Call ConnectObj.SelectFirm (1)
```

```
Set QryObj = ManagerObj.GetObjByName ("Sys", "ISysSqlQuery")
```

```
QryObj.Text = "select spr_nm from sspr where spr_rcd =: rcd"
```

```
Call QryObj.SetParamValue ("rcd", 1)
```

```
QryObj.OpenObj
```

```
QryObj.First
```

```
While QryObj.Next
```

```
If Not IsNull (QryObj.GetFieldValue ("Spr_Nm")) Then
```

```
MsgBox (QryObj.GetFieldValue ("Spr_Nm"))
```

End If

Wend

QryObj.CloseObj

Exit Sub

ErrorHandler:

MsgBox ("Error:" & Chr (13) & Err.Description)

End Sub

Опис прикладних COM-об'єктів **Комплексу**

1.4.10.2. Запуск ресурсних модулів з програми BASIC

В звітах Word і Excel для звернення до функцій **Комплексу** слід використовувати такі виклики:

RES_ADD_MODULE (<Найменування модуля>)

Додати модуль в список запускаються

RES_DEL_MODULES

Очистити список запускаються модулів

RES_LOAD_MODULES

Завантажити все модулі з списку

RES_UNLOAD_MODULES

Завершити роботи з модулями з списку

Приклад звернення до ресурсів **Комплексу** з звіту:

Sub ReportMain (Mgr As Object)

Set ComManager = Mgr

'Запросити інтерфейс зв'язку з **Комплексом**

```
Set Env = ComManager.GetObjByName ("Sys", "ISysEnvironment")
```

'Завантажити модуль

'Загальносистемні ресурси (SYS_R)

```
Env.RES_ADD_MODULE ("SYS_R")
```

```
Env.RES_LOAD_MODULES
```

'Вважати інформацію про контрагента

```
Env.PTN_READ_CD ("001")
```

'Завершити роботу з модулем

```
Env.RES_UNLOAD_MODULES
```

```
End Sub
```

Списки доступних модулів і їх вмісту надаються модулем доповнення до MS Office.

1.4.10.3. Проблеми при розробці звітів

Word і Excel звіти:

1. При створенні звіту Excel помилка

===== Помилка =====

Виклик методу: VBProject

Джерело: Microsoft Office Excel

опис:

Програмний доступ до проекту Visual Basic не є довіреною

Owner: SYSRXLT.CPP, File: SYSRXLT.CPP, Line: 654

=====

Для усунення помилки необхідно в програмі Excel за пунктом меню **Сервіс / Макрос / Безпека** на вкладці **Надійні видавці** зазначити параметр **Довіряти доступ до Visual Basic Project**.

2. При створенні звіту Word помилка

===== Помилка =====

Виклик методу: VBProject

Джерело: Microsoft

опис:

Відсутня довіра до програмованого доступу до проекту Visual Basic.

Owner: SYSRWRD.CPP, File: SYSRWRD.CPP, Line: 464

Для усунення помилки необхідно в програмі Word за пунктом меню **Сервіс / Макрос / Безпека** на вкладці **Надійні видавці** зазначити параметр **Довіряти доступ до Visual Basic**.

1.4.11. Використання в програмі користувача розрахункових функцій головної книги

УВАГА! НЕ ПІДТРИМУЮТЬСЯ:

розрахункові функції ACC_..., що діють в рамках функції ACC_CALC;

отримання оборотів і залишків TURN_..., BALANCE_...;

перебору по реєстру проводок ENT_GET_FCALC/ ENT_GET_NEXT !

Ці функції замінені розрахунковими функціями OSB_..., що діють в рамках функції OSB_CALC.

1.4.11.1. Програма отримання оборотів і сальдо по рахунку в програмах користувача

1. Перед заданням параметрів необхідно очистити буфер даних для підготовки розрахунку.

CALL

OSB_DATA_CLEAR()

- Наступним в програмі йде блок формування параметрів підготовки розрахунку. Оптимальним буде варіант, коли в цьому блоці декількома викликами функції **OSB_DATA** формується весь список даних для розрахунку.

Варіанти завдання параметрів розрахунку:

для отримання сальдо і оборотів по синтетичному рахунку (321):

CALL OSB_DATA("321",0,"",0)

Для отримання сальдо и оборотів детального рахунку (321/2) з вказаною аналітичною карткою (ID_AnI):

CALL OSB_DATA("321/2",ID_AnI,"",0)

для отримання оборотів по рахунку (321/2) в кореспонденції з рахунком (301):

CALL OSB_DATA("321/2",0,"301",0)

для отримання оборотів рахунка (321/2), враховуючи всі кор.рахунки:

CALL OSB_DATA("321/2",0,"*",0)

для отримання оборотів рахунка (321/2) в розрізі кор.рахунків та їх аналітики:

CALL OSB_DATA("321/2",0,"*",CONST_ULONG_MAX)

- Далі в програмі йде блок формування параметрів підготовки розрахунку. Оптимальним буде варіант, коли в цьому блоці декількома викликами функції **OSB_DATA** формується весь список даних для розрахунку.

Варіанти завдання параметрів розрахунку:

об'єкт за період I квартал, всі валюти:

Obj1= OSB_CALC("01/01/17";"31/03/17";"

об'єкт за період II квартал, всі валюти:

Obj2= OSB_CALC("01/04/17";"30/06/17";"

об'єкт за період I квартал у зазначеній валюті:

Obj3= OSB_CALC("01/01/17";"31/03/17";"USD")

Комплекс дозволяє підготувати одразу декілька об'єктів в розрізі різних періодів або валют.

Примітка – можливе використання розширеної функції **OSB_CALC2** для отримання розрахунків з додатковими параметрами.

- При викликанні першої функції отримання сальдо або оборотів, розраховується сальдо по рахункам в рамках вказаного об'єкту, підготованого **OSB_CALC** (див. п.п. 4.1.3). Наступні виклики розрахункових функцій отримують дані з буферу розрахунку.

Приклади функцій отримання вхідного сальдо:

VX_Dt1 =OSB_DT_BSLD(Obj1,"321/1", 0,"") ' вх.сальдо дебету (далі Дт) 321/1

VX_Dt2 =OSB_DT_BSLD(Obj2,"321/2,Id_An1,"") ' вх.сальдо Дт 321/2 і аналіт.карт.

VX_Kt1 =OSB_KT_BSLD(Obj1,"321/1",0,"") ' вх.сальдо кредиту (далі Кт) 321/1

VX_Kt2 =OSB_KT_BSLD(Obj2,"321/2,Id_An1,"") ' вх.сальдо Кт 321/2 и аналіт.карт.

Приклади функцій отримання вихідного сальдо:

IX_Dt1 = OSB_DT_ESLD (Obj1, "321/1", 0, "") ' вих.сальдо по дт 321/1

IX_Dt2=OSB_DT_ESLD(Obj2,"321/2",Id_An1,"") ' вих.сальдо по дт 321/2 і аналіт. карт.

Приклади функцій отримання оборотів:

OB_Dt1=OSB_DT_TURN(Obj1, "321/1", 0,"",0,"") ' обороти по дт 321/1

OB_Kt1=OSB_KT_TURN(Obj1, "321/1", 0,"",0,"") ' обороти по кт 321/1

OB_Dt2=OSB_DT_TURN(Obj2,"321/2",0,"301",0,"") ' обороти по дт 321/2, в кор. 301(см. правила заповнення **OSB_DATA** вище)

OB_Kt2=OSB_KT_TURN(Obj2, "321/2", 0,"301",ID_An12,"") ' обороти по дт 321/2, в кор. 301 з аналіт. ID_An12

(див. правила заповнення **OSB_DATA** вище)

1.4.11.2. Приклади використання розрахункових функцій в програмах користувача

- Для отримання сальдо и оборотів по дебету зазначеного рахунку 321

```
DATN=DATE_PACK(1,12,2017)
```

```
DATK=DATE_PACK(31,12,2017)
```

```
CALL OSB_DATA_CLEAR()
```

```
CALL OSB_DATA("321",0,"",0)
```

```
Obj= OSB_CALC(DATN,DATK,"")
```

```
OB_Dt =OSB_DT_TURN(Obj,"321", 0,"", 0, "")
```

```
OB_Dt1 =OSB_DT_TURN(Obj,"321/1", 0,"", 0, "")
```

```
OB_Dt2 =OSB_DT_TURN(Obj,"321/2", 0,"", 0, "")
```

- Для отримання сальдо по дебету и оборотів по дебету и кредиту рахунку 321/1 по аналітичній картці (ID- ІД аналіт. картки)

```
DATN=DATE_PACK(1,12,2017)
```

```
DATK=DATE_PACK(31,12,2017)
```

```
ID= ANLCRD_CREATE ("321/1","3507010","1111","",,,,,,,,"")
```

```
CALL OSB_DATA_CLEAR()
```

```
CALL OSB_DATA( "321/1", ID, "", 0 )
```

```
Obj=OSB_CALC(DATN, DATK,"")
```

```
VX_Dt=OSB_DT_BSLD( Obj, "321/1", ID, "" )
```

```
OB_Dt=OSB_DT_TURN( Obj, "321/1", ID, "", 0, "" )
```

```
OB_Kt=OSB_KT_TURN( Obj, "321/1", ID, "", 0, "" )
```

```
IX_Dt=OSB_DT_ESLD( Obj, "321/1", ID, "" )
```

- Для отримання оборотів по дебету 321/1 в кореспонденції с 701

```
DATN=DATE_PACK(1,12,2017)
```

```
DATK=DATE_PACK(31,12,2017)
```

```
CALL OSB_DATA_CLEAR()
```

```
CALL OSB_DATA("321/1",0,"701",0)
```

```
Obj=OSB_CALC(DATN, DATK,"")
```

```
OB_Dt =OSB_DT_TURN(Obj,"321/1", ID,"701", 0, "")
```

- Для отримання оборотів по дебету та кредиту 321/1 в розрізі кореспондуючих рахунків:

```
DATN=DATE_PACK(1,12,2017)
```

```
DATK=DATE_PACK(31,12,2017)
```

```
CALL OSB_DATA_CLEAR()
```

```
CALL OSB_DATA("321/1",0,"*",0)
```

```
Obj=OSB_CALC(DATN, DATK,"")
```

```
Korr_Sch = OSB_COR_FIRST (Obj, "321/1", 0 )
```

```
WHILE Korr_Sch <> ""
```

```
OB_Dt =OSB_DT_TURN(Obj,"321/1",0 , Korr_Sch,0,"" )
```

```
OB_Kt =OSB_KT_TURN(Obj,"321/1",0, Korr_Sch,0,"" )
```

```
Korr_Sch = OSB_COR_NEXT (Obj, "321/1",0)
```

```
WEND
```

- Для отримання оборотів по дебету і кредиту 321/1 в розрізі всіх аналітичних карток:

```
DATN=DATE_PACK(1,12,2017)
```

```
DATK=DATE_PACK(31,12,2017)
```

```
CALL OSB_DATA_CLEAR()
```

```
CALL OSB_DATA( "321/1",0,"",0)
```

```
Obj= OSB_CALC(DATN,DATK,"")
```

```
ID=
```

```
OSB_ANL_FIRST ( Obj, "321/1")
```

```
WHILE ID<>0
```

```
OB_DT =OSB_DT_TURN(Obj,"321/1",ID , "",0,"" )
```

```
OB_KT =OSB_KT_TURN(Obj,"321/1",ID , "",0,"" )
```

```
ID = OSB_ANL_NEXT (Obj,"321/1")
```

```
WEND
```

- Для отримання оборотів по дебету та кредиту 321/1 по аналітичним карткам зі вказаними реквізитами:

```
DATN=DATE_PACK(1,12,2017)
```

```
DATK=DATE_PACK(31,12,2017)
```

```
CALL OSB_DATA_CLEAR()
```

```
CALL OSB_DATA("321/1",0,"",0)
```

```
Obj= OSB_CALC(DATN,DATK,"")
```

```
ID = OSB_ANL_CDFST (Obj,0, "321/1","3507010", "", "", "", "", "", "", "", "", "", "")
```

```
WHILE ID <> 0
```

‘ ID — карточка последнего уровня картотеки

```

IF ANLCRD_GET_LEV(ID) = ANLCRD_LCNTSCH("321/1") THEN

OSB_Dt =OSB_DT_TURN(obj, "321/1", ID,"" , 0, "")

OSB_Kt =OSB_KT_TURN(obj, "321/1", ID,"" , 0, "")

END IF

ID = OSB_ANL_LNEXT (obj,1,"321/1")

WEND
    
```

- **Для отримання перебору проводок по рахунку**

```

DATN=DATE_PACK(1,12,2017)

DATK=DATE_PACK(31,12,2017)

CALL OSB_DATA_CLEAR()

CALL OSB_DATA("321/1",0,"",0)

Obj = OSB_CALC( DATN, DATK, "" )

CALL OSB_ENTRY_ON ( Obj )

ID_PRW = OSB_GET_FENTRY (Obj, 1)

WHILE ID_PRW <> 0

SCH_DT=ENT_DT(ID_PRW)

ANL_DT=ENT_DT_ANL(ID_PRW)

SCH_KT=ENT_KT(ID_PRW)

ANL_KT=ENT_KT_ANL(ID_PRW)

SUM_ PRW =ENT_SM(ID_PRW)

' ...

ID_PRW =
    
```

OSB_GET_NENTRY (Obj, 1)

WEND

- Для отримання перебору проводок по дебету рахунку

DATN=DATE_PACK(1,12,2017)

DATK=DATE_PACK(31,12,2017)

CALL OSB_DATA_CLEAR()

CALL OSB_DATA("321/1",0,"",0)

Obj = OSB_CALC(DATN,DATK, "")

CALL OSB_ENTRY_ON(Obj)

I=0

ID = OSB_ANL_FIRST(Obj,"321/1"

WHILE ID<>0

I=I+1

SUM_ANL =OSB_DT_TURN(Obj,"321/1", ID,"", 0, "")

IF SUM_ANL <>0 THEN

ID_PRW = **OSB_GET_FPENTRY** (Obj,I,"321/1", ID, "", 1)

DO WHILE ID_PRW <> 0

SUM_ PRW =ENT_SM(ID_PRW)

...

ID_PRW = OSB_GET_NENTRY(Obj,I)

LOOP

END IF

ID = OSB_ANL_NEXT(Obj,"321/1"

WEND

- Для отримання оборотів по дебету рахунку 321/1 по аналітичній картці (ID1), і зазначених кор.рахунків та їх аналітики

DATN=DATE_PACK(1,12,2017)

DATK=DATE_PACK(31,12,2017)

ID1= ANLCRD_ID ("321/1","3507010","1111",,,,,,,,,,,,,)

ID2= ANLCRD_ID ("701","3507010","1111",,,,,,,,,,,,,)

ID3= ANLCRD_ID ("301","1","3507010","00000001","1111",,,,,,,,,,,,,)

CALL OSB_DATA_CLEAR()

CALL OSB_DATA("321/1", ID1, "*", CONST_ULONG_MAX)

Obj=OSB_CALC(DATN, DATK,"")

OB_Dt1=OSB_DT_TURN(Obj, "321/1", ID1, "701", ID2 , "")

OB_Dt2=OSB_DT_TURN(Obj, "321/1", ID1, "301", ID3 , "")

1.4.11.3. Опис функцій головної книги

- Розрахункові функції по роботі с даними рахунків

OSB_DATA_CLEAR() – функція очищує буфер параметрів розрахунку.

OSB_DATA("Сч", Сч_АНЛ, "СчКОРР", СчКОРР_АНЛ) – функція добавляє в набір параметрів розрахунку вказані данні.

Параметри :

"Сч" - рахунок

Сч_АНЛ - аналітика рахунку

"СчКОРР" - кореспондуючий рахунок

СчКОРР_АНЛ - аналітика кореспондуючого рахунку.

Примітка – параметри "СчКОРР" та СчКОРР_АНЛ втратили своє призначення і не враховуються в зміненому алгоритмі функцій отримання сальдо и оборотів.

OSB_CALC(*Дата_Начала, Дата_Конца, "Вал"*) - функція готує об'єкт розрахунку відповідно до встановлених параметрів, періоду і валюти для фільтрації.

Параметри:

Дата_Начала - початок періоду розрахунків

Дата_Конца - кінець періоду розрахунку

"Валюта" - валюта фільтрації

Серед розрахункових функцій для роботи з даними рахунків добавлена функція «Подготовка Об'єкта расчета расширения»:

OSB_CALC2 (*Дата_Начала, Дата_Конца, "Валюта", Парам1, Парам2, Парам3, 0*)

Функція готує об'єкт розрахунку у відповідності з встановленими параметрами, періодом і валютою для фільтрації. У випадку вдалої підготовки вертається номер об'єкту. На основі цього номеру формують свої результати інші функції в цьому розділі.

Параметри:

Дата_Начала — початок періоду розрахунку

Дата_Конца — кінець періоду розрахунку

"Валюта" — валюта для фільтрації

Парам1 — отримання оборотів та сальдо по верхньому рівню аналітики

Парам2 — 0 - фактичний облік / 1 - плановий облік

Парам3 — 1 - Розрахунок в базовій валюті (Запит курсу на дату кінця розрахунку) 0 – додаткові параметри не задіяні

Примітка – Установка параметрів проводиться функцією **OSB_DATA**.

OSB_DT_BSLD(*Об'єкт, "Сч", Сч_АНЛ, "Вал"*)

Функція вертає значення вхідного залишку дебету по заданому рахунку та аналітиці, у зазначеній валюті.

Параметри:

Об'єкт - об'єкт, підготовлений *OSB_CALC*

"Сч" - рахунок

Сч_АНЛ - аналітика рахунку

"Вал" - валюта

Параметри Сч_АНЛ и *"Вал"* можуть не вказуватись. В параметрі *"Сч"* можна вказати консолідуючий або детальний рахунок.

Примітка – Для немультивалютних рахунків при встановленні параметра Валюта-"" або національна валюта результати виконання функцій **OSB_CALC**, **OSB_DT_BSLD**, **OSB_KT_BSLD**, **OSB_DT_TURN**, **OSB_KT_TURN** будуть однозначні. Для мультивалютних рахунків при встановленні параметра «Валюта» - іноземної валюти або національної валюти отримаємо сальдо та обороти рахунку в розрізі зазначеної валюти, при встановленні параметра «Валюта» - "" отримаємо сальдо та обороти в валюті обліку в цілому по рахунку.

OSB_KT_BSLD(*Об'єкт*, *"Сч"*, *Сч_АНЛ*, *"Вал"*) – функція вертає значення вхідного залишку кредиту по заданому рахунку і аналітиці, в зазначеній валюті.

OSB_DT_ESLD(*Об'єкт*, *"Сч"*, *Сч_АНЛ*, *"Вал"*) - функція вертає вихідний залишок дебету по зазначеному рахунку та аналітиці, в зазначеній валюті.

OSB_KT_ESLD(*Об'єкт*, *"Сч"*, *Сч_АНЛ*, *"Вал"*) - функція вертає вихідний залишок кредиту по зазначеному рахунку і аналітиці, в зазначеній валюті.

OSB_DT_TURN(*Об'єкт*, *"Сч"*, *Сч_АНЛ*, *"КорСч"*, *КорСч_АНЛ*, *"Вал"*)- функція вертає значення обороту по дебету рахунку в заданому співвідношенні.

Параметри :

Об'єкт - об'єкт, підготований *OSB_CALC*

"Сч" - рахунок

Сч_АНЛ - аналітика рахунку

"КорСч" - кор.рахунок

КорСч_АНЛ - аналітика кор.рахунку

"Вал" - валюта

Параметри Сч_АНЛ, "КорСч", КорСч_АНЛ, "Вал" можуть не вказуватись або

вказуються частково. В параметрі "Сч" можна зазначити консолідуєчий або детальний рахунок, а в параметрі "КорСч" – тільки детальний рахунок.

OSB_KT_TURN(*Об'єкт, "Сч", Сч_АНЛ, "КорСч", КорСч_АНЛ, "Вал"*) - функція вертає значення обороту по кредиту .

OSB_ENTRY_ON(*Об'єкт*) - Функція встановлює режим збору даних о проводках.

Об'єкт — об'єкт, підготований OSB_CALC.

Режим необхідний для роботи функцій OSB_GET_FENTRY и OSB_GET_NENTRY.

OSB_GET_FENTRY(*Об'єкт, Номер_Цикла*) - Функція вертає ID першої проводки розрахунку по зазначеним параметрам.

Параметри:

Об'єкт - об'єкт, підготований **OSB_CALC**;

Номер_Цикла - номер циклу ітерації.

Примітка –Функція працює тільки у випадку, коли для об'єкту розрахунку був встановлений режим збору інформації о проводках функцією **OSB_ENTRY_ON**.

OSB_GET_NENTRY(*ID_Об'єкта, Номер_Цикла*) - Функція вертає ID наступної проводки розрахунку за зазначеними параметрами.

Параметри:

Об'єкт - об'єкт, підготований **OSB_CALC**;

Номер_Цикла - номер циклу ітерації.

Примітка –Функція працює тільки в тому випадку, коли для об'єкту розрахунку був встановлений режим збору інформації о проводках функцією **OSB_ENTRY_ON**.

OSB_COR_FIRST_ALL(Об'єкт, "Счет", Сч_АНЛ); OSB_COR_NEXT_ALL(..);

OSB_COR_LFIRST_ALL(Об'єкт, Номер_Цикла, "Сч", Сч_АНЛ); OSB_COR_LNEXT_ALL(...),

котрі відрізняються від вже наявних в цьому розділі тим, що в параметр «Сч» може передаватися як детальні, так і консолідуючі бухгалтерські субрахунки.

OSB_GET_FPENTRY(Об'єкт, Номер_Цикла, "Счет", Сч_Анл, "Корр_Счет", Тип_Обор),

Функція вертає ID першої проводки, відібраної останньою функцією **ACC_CALC**, що задовольняє зазначеним параметрам. Функція працює лише в тому випадку, коли для об'єкту розрахунку **OSB** був встановлений режим збору інформації о проводках функцією **OSB_ENTRY_ON**.

Об'єкт - об'єкт, підготований OSB_CALC,

Номер_Цикла - номер циклу ітерації,

"Счет" - рахунок фільтрації (обов'язковий параметр),

Сч_Анл - аналітика рахунку фільтрації,

«Корр_Счет» - кореспондуючий рахунок фільтрації,

Тип_Оборота - тип оборотів (1 - по дебету, 2 - по кредиту, 0 – по дебету і кредиту).

- Підгрупа функцій для отримання кореспондуючих рахунків:

OSB_COR_FIRST(Об'єкт, "Счет", Сч_АНЛ)

Функція вертає перший кор.рахунок за зазначеними параметрами в рамках об'єкту, підготованого **OSB_CALC**.

OSB_COR_LFIRST(Об'єкт, Номер_Цикла, "Сч", Сч_АНЛ)

Функція повертає перший кор.рахунок за зазначеними параметрами з урахуванням циклу сканування та в рамках об'єкту, підготованого **OSB_CALC**.

OSB_COR_NEXT(Об'єкт, "Счет", Сч_АНЛ) - функція вертає наступний кор.рахунок за вказаними параметрами в рамках об'єкту, підготованого **OSB_CALC**.

OSB_COR_LNEXT(Об'єкт, Номер_Цикла, "Сч", Сч_АНЛ)

Функція вертає наступний кор.рахунок за зазначеними параметрами з урахуванням циклу сканування та в рамках об'єкту, підготованого **OSB_CALC**.

- Підгрупа функцій для перебору аналітичних карток рахунку:

OSB_ANL_FIRST(*Об'єкт*, "Сч")

Функція повертає першу аналітичну картку рахунку, по якій є обороти або сальдо в рамках об'єкту, підготованого OSB_CALC.

OSB_ANL_LFIRST(*Об'єкт*, *Номер_Цикла*, "Сч")

Функція повертає першу аналітичну картку рахунку, по якій є обороти або сальдо в рамках об'єкту, підготованого OSB_CALC, з урахуванням циклу сканування.

OSB_ANL_NEXT(*Об'єкт*, "Сч")

Функція повертає наступну аналітичну картку рахунку, по якій є обороти або сальдо в рамках об'єкту, підготованого OSB_CALC.

OSB_ANL_LNEXT(*Об'єкт*, *Номер_цикла*, "Сч")

Функція повертає наступну аналітичну картку рахунку, по якій є обороти або сальдо в рамках об'єкта, підготованого OSB_CALC, з урахуванням циклу сканування.

OSB_ANL_CDFST(*Об'єкт*, *Номер_Цикла*, "Сч", "*Код1*", "*Код2*", "*Код3*", "*Код4*", "*Код5*", "*Код6*", "*Код7*", "*Код8*", "*Код9*", "*Код10*")

Функція повертає ID першої аналітичної картки на рахунку, по якій є обороти або сальдо в рамках об'єкту, підготованого OSB_CALC, відповідній переданим кодам. Код аналітики на кожному рівні може бути заповнений або не заповнений (""). Наступне значення можна отримати за допомогою функції OSB_ANL_LNEXT.

OSB_ANL_LVLFST(*Об'єкт*, *Номер_Цикла*, "Сч", *НомУровня*)

Функція повертає ID першої аналітичної картки на рахунку, по якій є обороти або сальдо в рамках об'єкту, підготованого OSB_CALC на вказаному рівні, з вкладеним циклом. Наступне значення можна отримати за допомогою функції OSB_ANL_LNEXT.

- Функции доступа к параметрам проводки

ENT_DATE (ID_проводки)

Функція повертає дату проводки по заданому ID.

ENT_DT (ID_проводки)

Функція повертає номер рахунку дебету проводки по заданому ID.

ENT_DT_ANL (ID_проводки)

Функція повертає ID аналітичної карточки по рахунку дебету проводки із заданим ID.

ENT_KT (ID_проводки)

Функція повертає номер рахунку кредиту проводки по заданому ID.

ENT_KT_ANL (ID_проводки)

Функція повертає ID аналітичної картки по рахунку кредиту проводки із заданим ID.

ENT_NOTE (ID_проводки)

Функція повертає рядок коментаря до проводки із заданим ID.

ENT_OPR_DOC (ID_проводки)

Функція повертає код документа господарської операції, до якої прив'язана проводка із заданим ID.

ENT_OPR_DOCID (ID_проводки).

Функція повертає ID документа господарської операції, до якої прив'язана проводка із заданим ID.

ENT_OPR_DOCNMV (ID_проводки)

Функція отримання зовнішнього номера документа господарської операції, до якої прив'язана проводка.

ENT_OPR_DOCDAT (ID_проводки)

Функція повертає дату документа господарської операції, до якої прив'язана проводка із заданим ID.

ENT_OPR_DOCNMR (ID_проводки)

Функція повертає номер документа господарської операції, до якої прив'язана проводка із заданим ID.

ENT_OPR_ID (ID_проводки)

Функція повертає ID господарської операції, до якої прив'язана проводка із заданим ID.

ENT_OPR_NM (ID_проводки)

Функція повертає найменування господарської операції, до якої прив'язана проводка із заданим ID.

ENT_OPR_TO (ID_проводки)

Функція повертає ID типової операції (шаблону проводок), на підставі якої створена господарська операція проводки.

ENT_POST_CD (ID_проводки)

Функція повертає ідентифікатор користувача проводки із заданим ID.

ENT_PRZ_HND (ID_проводки)

Функція отримує ознаку ручної проводки по заданому ID. Функція повертає 1, якщо проводка корегувалася користувачем, 0 – в іншому випадку.

ENT_SM (ID_проводки)

Функція повертає проводки по заданому ID.

ENT_VAL (ID_проводки)

Функція повертає валюту проводки по заданому ID.

ENT_VALCUR (ID_проводки)

Функція повертає курс валюти проводки по заданому ID.

ENT_USER (ID_проводки)

Функція повертає ім'я користувача, який провів операцію з проводками в Головний журнал по ID проводки.

Показчик

к

Конструктор

4

п

Програмування в системі

4