Среда
разработки и выполнения приложений
«Live Universal Interface»

Руководство программиста

Оглавление

Введение

Описываемый программный продукт “Live Universal Interface” (LUI) родился и развивался в недрах большой, долгоживущей, тиражной системы для автоматизации деятельности предприятий связи – АСР Fastcom.

Нас, разработчиков этой Автоматизированной Системы Расчётов, не мог не беспокоить, прежде всего, вопрос конструирования и применения единого принципа реализации интерфейса пользователя. Самое естественное было бы выбрать самую подходящую из имеющихся на рынке систем построения интерфейса. В большинстве случаев так и поступают – выбирают систему, с помощью которой быстрее всего достигаются поставленные цели. Но в этих случаях цель, как правило, выполнение одного конкретного контракта. В нашем случае цель была гораздо шире: Выполнение множества контрактов с разнообразнейшими требованиями, меняющимися со временем согласно конъюнктуре бизнеса и изменению конкурентной среды. Если конкретизировать эти требования, то получится следующее:

1. Интерфейс должен легко адаптироваться для отображения на любых существующих и появляющихся в будущем интерактивных системах взаимодействия с пользователем;
2. Интерфейс должен быть гибким, легко настраиваемым под потребности конкретного заказчика и, более того, настраиваемым под меняющиеся со временем потребности конкретного заказчика;
3. Интерфейс должен быть адаптивным к данным, то есть уметь автоматически изменяться в зависимости от отображаемых данных;
4. Разработчики бизнес-логики должны пользоваться таким инструментом построения интерфейсных форм, который не зависел бы от системы взаимодействия с пользователем.

Основная идея, проходящая красной нитью через всю концепцию LUI – это чёткое разграничение деятельности программиста прикладной системы и разработчика средства отображения интерфейса («движок»). Второй интерпретирует результаты деятельности первого. Обмен происходит на уровне абстрактных понятий, универсальных для любого средства разработки и отображения интерфейса. Разработчик прикладной системы только обозначает, какую функциональность он ожидает от интерфейса. Как и какими средствами эта функциональность будет реализована – не его забота. Таким образом, разработка прикладной системы оказывается полностью отделённой от средства (среды и языка) разработки «движка» или нескольких «движков», которыми осуществляется и будет осуществляться в будущем интерпретация труда программиста.

Как же достигается поставленная цель?

Первое – как уже сказано, абстрагирование свойств элементов интерфейса. Для примера: чтобы визуально выделить элемент среди других элементов интерфейса, разработчик прикладной системы не оперирует такими свойствами, как шрифт (вдруг «движок» алфавитно-цифровой?), цвет (вдруг «движок» монохромный?), фон и т.п. Он оперирует ролями, о которых существует «договорённость» с разработчиком «движка»: «Внимание», «Опасность», «Незаметный» и т.п. А уже разработчик конкретного движка на свой вкус и, исходя их предоставленных ему инструментария, интерпретирует эти роли.

При существенных расхождениях во мнении об интерфейсных реализациях свойств элементов интерфейса разработчик «движка» реализует средства кастомизации (настройки) под конкретный Проект или даже под любого Пользователя внутри Проекта.

Второе – это уход при разработке прикладного кода от понятия «Событие». При описании интерфейса нельзя мыслить в категории событий и триггеров, так как они являются очень специфическими для конкретного «движка», разработанного в конкретной среде. Продвинутые средства поддерживают множество типов событий, а другие – не так много, но зато берут обработку части событий на себя. В одних средах под одним и тем же событием подразумевается одно, а в других – другое. Одни считают, что при каком-то событии надо произвести такой набор стандартных действий, а другие – совсем иной.

Поэтому, в LUI многообразие отношений интерфейсных элементов реализуется не описанием обработчиков событий, а декларативными динамическими свойствами. То есть свойства элементов интерфейса могут изменяться как при синтезе интерфейсных форм движком, так и уже в процессе их функционирования – в зависимости от входных параметров, значений контекстных элементов, параметров контента, изменяемых пользователем элементов, а также результатов запроса к прикладной системе (бизнес-логика).

Для реализации данной концепции вполне достаточно декларативного языка – SQL, но также подойдут и многие процедурные языки – PLpgSQL,PL/SQL, JavaScript и т.п. В LUI описание динамических свойств производится так:

Динамикой являются конструкции вида {Язык:Команда}. Их наличие – указание интерпретатору, что в некоторый момент (какой – решает «движок») содержание фигурных скобок должно быть проинтерпретировано, вычислено и заменено (включая и сами скобки) на результат вычисления.

Пример динамического свойства – заголовка таблички с объектами договора:

```
Объекты договор{SQL:select COALESCE(MAX('а №'||CONTR_NO), 'ов') from contracts where
ID=0{Param:CONTRID}}
```

Это свойство содержит динамический фрагмент в виде запроса к базе данных, опирающийся на входной параметр CONTRID. Ноль перед параметром страхует от отсутствия значения параметра CONTRID.

Динамические фрагменты могут быть любой глубины вложенности друг в друга. Определение момента вычисления и перевычисления динамических фрагментов – задача «движка». Отслеживание зависимостей динамических фрагментов от ссылок на меняющиеся элементы – тоже задача разработчика «движка». Разработка «движка» - вообще, довольно сложная работа, так как с одной стороны нельзя упустить момент изменения данных, от которых зависят значения с динамическими фрагментами, с другой стороны необходимо минимизировать трафик от сервера к клиенту и не делать лишних перевычислений динамических фрагментов.

Базовые принципы LUI

1. Алгоритмы изменений интерфейсных элементов в процессе работы форм описываются декларациями динамических свойств элементов форм, а не процедурами обработки событий в формах. Движок обеспечивает адекватность отображения текущего значения свойства каждого элемента в любой момент времени.
2. Набор свойств элемента фиксирован и согласован. То есть, не должно быть таких свойств элемента, значения которых могут противоречить друг другу или быть несовместимыми.
3. Элементы и их свойства должны быть достаточно абстрактными, чтобы не привязываться к конкретной реализации интерфейсного движка. Иными словами, свойства оперируют понятиями, а не физическими устройствами.
4. Программы на языке, обеспечивающем декларации динамических фрагментов, не должны возбуждать необработанных исключений и ошибок. В любой момент времени вычисление динамического фрагмента должно возвращать какое-то значение без возникновения ошибки.
5. Динамические фрагменты должны вычисляться только по мере необходимости и только непосредственно в момент их востребованности.
6. При обработке клиентского события единожды вычисленный динамический фрагмент не перевычисляется при повторных обращениях до тех пор, пока не изменятся элементы, использующиеся прямо или косвенно при вычислении.
7. Движок при реакции на событие должен передавать на клиентское устройство команды по изменению интерфейсных элементов, минимизируя трафик – то есть, отправляться должны только новые или реально изменившиеся свойства.

Концепции LUI

Верхней структурной единицей в LUI является **Прикладная Система (Приложение)**. Приложения уникально идентифицируются алфавитно-цифровым кодом. Каждое приложение состоит из форм. В рамках приложения каждая форма имеет уникальный алфавитно-цифровой идентификатор – код формы. Описания приложений, форм и их элементов хранятся в таблицах базы данных. Совокупность этих описаний называют **метаданными**. В процессе работы прикладной системы эти описания считываются сервером приложений и интерпретируются для отображения на экране пользователя и реакции на действия пользователя клавиатурой, мышью или иными средствами, поддерживаемыми конечным оборудованием пользователя. Прикладные системы, формы и их элементы создаются и изменяются в среде разработке приложений редактором форм. Среда разработки приложений и редактор форм тоже являются прикладной системой, которая имеет код **LUI**.

LUI умеет отображать формы следующих типов:

- ***Списки***;
- ***Бланки***;
- ***Иерархические структуры на основе списков***
- ***Иерархические меню***

Кроме того, сервер приложений умеет обрабатывать не интерфейсные формы следующих типов:

- ***Команды ОС***;
- ***Задачи (условно длительно выполняемые рутинные процедуры)***;
- ***Верификаторы данных***

Формы могут принимать набор ***Входных параметров***, которые влияют на внешний вид и логику работы формы. Для входных параметров можно указывать значение по умолчанию, которое будет присвоено параметру, если вызывающая (родительская) форма при вызове не укажет значение этого параметра.

Каждая интерфейсная форма содержит набор визуальных элементов:

- ***Строк списка***;
- ***Заголовков столбцов***;
- ***Ячеек строк***;
- ***Полей ввода*** (разного типа);
- ***Кнопочек***;
- ***Иконок;***
- ***Пунктов меню;***

Все визуальные элементы интерфейсных форм могут группироваться при помощи специальных элементов - ***Групп***, которые, впрочем, тоже могут группироваться в Группы более высокого уровня.

В общем случае каждая интерфейсная форма может активировать ***Действия*** различными способами:

- Клик левой клавишей мыши на интерфейсной кнопочке;
- Клик левой клавишей мыши на иконке;
- Выбор пункта ***Локального меню***;
- Ролевая клавиша клавиатуры: ***Enter***, ***Insert***, ***Delete***;
- Двойной клик левой клавишей мыши;
- Попытка завершения работы формы;
- Тик таймера;
- Автоматическое выполнение по клиентскому событию.

Посредством действий может не только выполняться функционал бизнес-логики, но и вызываться дочерние формы. Все вызываемые формы открываются в модальном (по отношению к вызывающей форме) режиме – то есть, пока дочерняя форма не завершится, в родительской форме не обрабатываются никакие события. Исключение – формы, открываемые автоматически по событию. В этом случае форма открывается в том же окне не в эксклюзивном режиме – то есть, в вызывающей форме всё ещё обрабатываются любые события.

Действия могут иметь набор ***Выходных параметров***, значения которых становятся значениями входных параметров вызываемой формы.

В иерархических структурах Действия, вызывающие Списки, могут порождать подчинённые (вложенные) элементы иерархической структуры. Именно таким образом в иерархических структурах возникает, собственно, иерархия.

Любой описываемый элемент формы (столбец, поле, группа, действие, параметр), так же, как и сама форма, имеют специфический набор ***Свойств***, совокупное значение которых определяет внешний вид и способ функционирования формы. Все элементы формы имеют уникальный в пределах формы алфавитно-цифровой идентификатор – код элемента.

Все наборы заданных значений свойств одной формы могут объединяться в некоторой ***Конфигурации***. Конфигурации формы могут выстраиваться в иерархию. Каждая конфигурация в рамках формы имеет уникальный алфавитно-цифровой код. При этом у любой формы должна обязательно быть одна конфигурация верхнего уровня с кодом DEFAULT. Неопределённые значения свойств в некоторой подчинённой конфигурации наследуют значение этого свойства у родительской конфигурации, а при её отсутствии – принимают значение свойства по умолчанию.

Значения свойств могут быть:

- Статическими
- Динамическими

В первом случае значение свойства не меняется на протяжении всего функционирования формы, во втором случае значение свойства меняется в соответствии с декларацией и всегда поддерживается сервером приложений в актуальном состоянии.

Динамическое значение свойства элемента, помимо возможной статической части, содержит или включает в себя минимум одну динамическую составляющую (***Динамический элемент***). Динамический элемент это конструкция на языке LUI. Особенностью языка LUI является то, что он лишь предназначен для обращения к другим языкам. Иногда его называют метаязыком.

Динамический элемент является оператором этого метаязыка и определяется шаблоном:

```
{КодЯзыка:ЭлементыЯзыка}
```

Динамические элементы могут вкладываться друг в друга. В этом случае вычисление (***парсинг*** динамики) идёт, начиная с вложенных элементов. Порядок вычисления значений динамических элементов одного уровня (рядом расположенных) – не определён и не гарантируется. Результатом вычисления динамического элемента может быть другой динамический элемент, но по умолчанию он уже не будет вычислен (рекурсии нет).

Динамические элементы значений свойств формы вычисляются тогда и только в тот момент, когда они востребованы сервером приложений. Если свойство действует постоянно (например, отображение текста заголовка формы), то перевычисление динамического значения будет производиться всегда в тот момент, когда и если изменятся опорные элементы, которые использовались при последнем парсинге динамического элемента (значения ячеек, полей, свойств и переменных формы).

Для реализации обращений к различным языкам в сервере приложений применена технология подключаемых модулей (plugin). Каждый подключаемый модуль реализует обращение к одному языку и обеспечивает вычисление динамического элемента на этом языке. Например, плагин SQL обеспечивает вычисление путём запроса к базе данных.

Описание языков динамических элементов свойств

★ F – значение элемента формы

{**F**:***КодЭлемента***}

Синонимы:

{**Field**:***КодПоля***}

{**Column**:***КодСтолбца***}

{**Param**:***КодПараметра***}

Возвращает вместо себя значение элемента с указанным кодом из текущей формы. Элементы – это могут быть поля бланков, ячейки текущей строки списков и входные параметры формы. Элемент должен су-

ществовать в форме на момент первого вычисления данного динамического выражения. Если такового не найдётся, то выражение не будет вычислено – останется как есть вместе с фигурными скобками.

Примеры:

- {F:CODE} – заменится на текущее значение поля CODE
- {F:IN_PARAM} – заменится на значение входного параметра IN_PARAM
- {F:{Property:Var CurrentField}} – Заменится на текущее значение текущего поля бланка

★ F2SQL - значение элемента формы сконвертированное для языка SQL

{**F2SQL**:***КодЭлемента***}

Возвращает вместо себя значение элемента с указанным кодом из текущей формы, причём значение будет преобразовано для применения внутри операторов языка SQL целевой СУБД. Элементы – это могут быть поля бланков, ячейки текущей строки списков и входные параметры формы. Элемент должен существовать в форме на момент первого вычисления данного динамического выражения. Если такового не найдётся, то выражение не будет вычислено – останется как есть вместе с фигурными скобками. “F2SQL “ упрощает написание выражений на SQL по сравнению с “F” . Например, пусть в форме существует поле с кодом DT типа DATETIME и форматной маской “dd.mm.yyyy hh24:mi:ss”, и пусть существует таблица MY_TABLE со столбцом DT типа TIMESTAMP.

Тогда оператор INSERT, если использовать “F”, надо будет кодировать так:

insert into my_table(dt) values(to_timestamp(‘{F:DT}’,’dd.mm.yyyy hh24:mi:ss’));

В то время как, оператор INSERT если использовать “F2SQL” будет выглядеть так:

insert into my_table(dt) values({F2SQL:DT});

Если в поле DT введено “17.07.2019 18:10:23”, то в результате вычисления выражения “{F2SQL:DT}” получим:

insert into my_table(dt) values(to_timestamp(‘17.07.2019 18:10:23’,’dd.mm.yyyy hh24:mi:ss’));

Таким образом, очевидно преимущество применения F2SQL для кодирования выражений отдаваемых языку SQL.

★ Property – свойство текущего элемента (каскад)

{**Property**:***КодСвойства***}

Синоним:

{**P**:***КодСвойства***}

Возвращает значение указанного свойства текущего элемента. Если у текущего элемента указанного свойства не найдутся, то оно ищется у родительских элементов вверх по иерархии вплоть до формы. Если и там нет, то свойство ищется у приложения. Если указанного свойства не найдётся нигде, то выражение не будет вычислено – останется как есть вместе с Фигурными скобками.

Примеры:

- `{Property:FormID}` – заменится на ID текущего экземпляра формы
- `{P:UserChanged}` – у поля или в подчинённых локальных действиях вернёт Y или N – в зависимости от того – было ли изменено значение поля непосредственно пользователем

★ ElementProperty – свойство элемента формы

{**ElementProperty**:***КодЭлемента*.*КодСвойства***}

Возвращает значение указанного свойства указанного элемента текущей формы. Если в форме нет указанного элемента или у него нет такого свойства, то выражение не будет вычислено – останется как есть вместе с Фигурными скобками.

Примеры:

- `{ElementProperty:MULTILANG.Parent}` – возвращает код родительского элемента действия MULTILANG.

- `{ElementProperty:RUN.LastRunSCN}` – возвращает уникальный идентификатор вызова действия RUN.

★ FormProperty – параметры и переменные формы

{**FormProperty**:***КодСвойства***}

Возвращает значение указанного свойства текущей формы. Если такого свойства не обнаружится, то оно будет создано в данном экземпляре формы с пустым значением (не null) и вернёт его. При этом будет построена зависимость и при обновлении этого свойства в будущем зависимые свойства будут инвалидироваться (т.е. считаться требующими вычисления) для перевычисления по первому требованию.

Так как значения свойств формы живут только до конца жизни текущего экземпляра формы, то этот динамический элемент является удобным инструментом для создания программистами переменных уровня экземпляра формы.

Примеры:

- `{FormProperty:CountSelected}` – в формах-списках возвращает количество выделенных строк.
- `{FormProperty:Var CurrentField}` – в формах-бланках возвращает код текущего поля.
- `{FormProperty:MyNewProperty}` – обращение к пользовательской переменной MyNewProperty

★ ApplicationProperty – глобальные параметры и переменные уровня сеанса в прикладной системе

{**ApplicationProperty**:***КодСвойства***}

Возвращает значение указанного свойства приложения. Если такого свойства не обнаружится, то оно будет создано в данном сеансе приложения с пустым значением (не null) и вернёт его.

Так как значения свойств сеанса живут только до конца жизни текущего сеанса приложения, то этот динамический элемент является удобным инструментом для создания программистами переменных уровня текущего пользовательского сеанса.

Примеры:

- `{ApplicationProperty:CURRENT_LANG}` – код языка, в котором сейчас работает сессия.
- `{ApplicationProperty:SessionID}` – уникальный идентификатор текущего сеанса
- `{ApplicationProperty:AppCode}` – код текущего приложения

★ SQL – запрос к БД

{**SQL**:***Запрос***}

{**SQL**:(***КодСоединения***)***Запрос***}

Заменяется конкатенацией всех строк первого столбца запроса. Возвращает пустую строку, если запрос ничего не вернёт или если это вообще не запрос, а, например, insert или commit.

Примеры:

- `{SQL:select translate('{F:NAME}', '0123456789', 'ABCDEFGHIJ')}`
  – заменяется значением поля/столбца/параметра NAME, где все цифры подменены прописными буквами.
- `{SQL:select coalesce(max('Y'), 'N') from lui_t_language where code='{Param:LANG_CODE}'}`
  – заменяется на Y, если значение входного параметра LANG_CODE есть в таблице и N – если нет.
- `{SQL:select initcap(Code) from lui_t_formtype order by seq}`
  – заменяется на строку ListBlankMenuHostChart

(КодСоединения) – это указание на то, какое именно соединение следует использовать при обращении к БД. Сеанс работы в прикладной системе может одновременно иметь несколько соединений с разными БД. Если код соединения не указан явно, то

★ JavaScript – вычисление выражения на языке Java Script.

{**JavaScript**:***ПрограммныйКод***}

Синоним:

{**JS**:***ПрограммныйКод***}

Программный код тут воспринимается как выражение, возвращающее значение. Это значение и заменит динамическую конструкцию при вычислении.

Примеры:

- `{JS:"{Property:SelectionMode}"==="Single"?"EXIST":"N"}`
  – заменяется на `EXIST`, если сейчас режим выделения одной строки, а иначе на `N`
- `{JS:"{Param:EDIT_CONFIG_CODE}"?"Y":"N"}`
  – если входной параметр `EDIT_CONFIG_CODE` задан – заменится на Y, иначе – на N.
- `{JS:if ("{FormProperty:SelectionMode}"==="Multi")`
  `"Удаляемых строк - {FormProperty:CountSelected}";`
  `else "Удалить строку с кодом {F:CODE}?";}`
  – заменяется на текст вопроса, различающийся в зависимости от режима выделения строк

★ NLS – мультиязычность

{**NLS**:***КодЯзыка1***:***Текст1***;… ***КодЯзыкаN***:***ТекстN***;}

{**NLS**:[***КодЯзыка***]***КодЯзыка1***:***Текст1***;… ***КодЯзыкаN***:***ТекстN***;}

Заменяется текстом с номером того языка, который сейчас установлен в сеансе. Если текста на нужном языке нет, то будет искаться текст на замещающем языке и т.д.

Возможно указание в квадратных скобках предпочтительного (взамен текущего) языка.

Примеры:

- `{NLS:ENG:Table;RUS:Таблица;}` – заменяется на «`Таблица`»,
  если глобальный параметр `{ApplicationProperty:CURRENT_LANG} = RUS`
- `{NLS:[{Param:IN_LANG}]ENG:{F:CODE}-Table;RUS:Таблица {F:CODE};}`
  – заменяется текстом на том языке, который указан во входном параметре `IN_LANG`

★ LOVC – выбор из вариантов (коллекции)

Выберите:	
1	Один
2	Два
N	Много
	Ничего

{**LOVC**:***ЗаголовокВыбора***;***Выбор1***:***Текст1***;… ***ВыборN***:***ТекстN***;}

(List Of Values from Collection). При вычислении свойства с такой конструкцией вычислитель выражений LUI приостанавливает свою работу, а для пользователя отображается модальное окно со списком из текстов от 1 до N. Текст заголовка выбора станет заголовком этого модального окна.

Пользователь делает выбор ТекстаM и вся динамическая конструкция заменяется на ВыборM. После этого вычисление выражения продолжается. Если пользователь отказывается от выбора – вычисление прерывается, выполняется отказ от действия и делается попытка отката на момент начала действия, породившего этот список значений.

Примеры:

- `{LOVC:1:Один;2:Два;N:Много;:Ничего;}` – пользователю отобразится список вариантов и, в зависимости от его выбора, конструкция заменится на 1, 2, N или пусто.
- `SQL:{LOVC:Данные подготовлены;commit:Сохранить;rollback:Отменить;}` – в зависимости от выбора пользователя произойдёт или фиксация результатов работы в базе, или откат к началу транзакции.

★ LOVD – выбор из выпадающего списка (коллекции)

{**LOVD**:***Выбор1***:***Текст1***;… ***ВыборN***:***ТекстN***;}

(List Of Values from Dropdown). Используется в свойствах, вычисляющихся в контексте некоего поля. При вычислении свойства с такой конструкцией вычислитель выражений LUI приостанавливает свою работу, а клиенту показывают выпадающий из текущего поля список текстов от 1 до N.

Пользователь делает выбор ТекстаМ и вся динамическая конструкция заменяется на ВыборМ. После этого вычисление продолжается. Если пользователь отказывается от выбора – вычисление прерывается, выполняется отказ от действия и делается попытка отката на момент начала действия, породившего этот LOV.

Значения Выбор1… ВыборN пользователю не видны!

Примеры:

См. LOVC – выбор из вариантов (коллекции) на стр. 11

★ LOVQ – Выбор значений из запроса

{**LOVQ**:***Запрос***}

{**LOVQ**:(***КодСоединения***)***Запрос***}

(КодСоединения) – это указание на то, какое именно соединение следует использовать при обращении к БД. Сеанс работы в прикладной системе может одновременно иметь несколько соединений с разными БД.

(List Of Values from Query). При вычислении свойства с такой конструкцией вычислитель выражений приостанавливает свою работу, а клиенту в отдельном модальном окошке показывают список, сформированный из результата выполнения запроса. Каждый столбец запроса порождает столбец в окошке LOV.

Если хотя бы у одного столбца есть текстовый псевдоним - alias(в двойных кавычках), то видимость столбцов будет управляться вручную – скрываются столбцы без псевдонима. Если псевдонимов нет вообще, то видимость столбцов вычисляется автоматически: скрываются столбцы с типом данных number. А заголовков у столбцов не будет вообще.

Пользователь может выбрать любую строку. В этом случае результатом вычисления LOV станет значение первого столбца выбранной строки. Не имеет значения – видим ли первый столбец или нет. После выбора вычисление значения продолжается. Если пользователь отказывается от выбора – вычисление прерывается, выполняется отказ от действия и делается попытка отката на момент начала действия, породившего этот LOV.

- Столбец с псевдонимом ROW_STATUS имеет отдельную функциональность. Прежде всего, этот столбец не показывается пользователю. Значение этого столбца определяет тип всей строки LOV.
- Значение 0 говорит о том, что строка чисто информационная – её нельзя выбрать. Такая строка используется для заголовков разделов, временно деактивированных значений и т.п.
- Значение 2 указывает на то, что строку нельзя выбрать, то её можно «раскрыть» - у неё появляется значок, клик на котором даёт системе команду отобразить подчинённые строки.
- Значение 3 – комбинированная строка. Её можно раскрыть, но её же можно и выбрать.
- Любые другие значения, равно как и отсутствие столбца ROW_STATUS, порождают обычные строки LOV, которые можно выбрать, но нельзя раскрыть.

В иерархических LOV-ах запрос должен содержать переменную {PARENT_NODE}. Для отображения строк первого уровня эта переменная заполняется пустым значением. В случае попытки раскрытия узла, эта переменная заполняется значением первого столбца раскрываемого элемента, и запрос LOV выполнятся ещё раз для генерации вложенных элементов.

Заголовок окна LOV имеет стандартный текст: «Выберите» или «Choose». Но разработчик может его заменить. Для этого в тексте LOV (например, внутри комментария) должна присутствовать конструкция:

`[TITLE:`***ТекстЗаголовка***`]`

Примеры:

- ```
  {LOVQ:select user_name "{NLS:RUS:Пользователь;ENG:User;'}"
   from lui_t_user where app_code='{F:P_APPCODE}'}
  ```
  ```

– отображает пользователю LOV из одного столбца со списком всех пользователей приложения, указанного в параметре `P_APPCODE`.

- ```
  {LOVQ:Select /*[TITLE:{NLS:RUS:Добавить ссылку;ENG:add reference;}]*/ as code,
  nls(name, '{Property:CURRENT_LANG}') as name
  from lui_t_element where app_code='{F:APPLICATION_CODE}'
  and form_code='{F:FORM_CODE}'
  and etype='PARAM'
  and element_code is null}
  ```
  – Выбирает из метаданных и отображает список входных параметров формы `FORM_CODE` приложения `APPLICATION_CODE`.

★ PLPGSQL – вычисление выражения языке PL/pgSQL

{**PLPGSQL**:(***КодСоединения***)***ПрограммныйКод***}

Код соединения в круглых скобках можно не указывать, если соединение с базой данных только одно.

Для того, чтобы процедурный язык PL/pgSQL можно было использовать как декларации динамических свойств, внутри кода как минимум один раз должна выполняться установка параметра сеанса “LUI.RETVAL”. Это можно сделать с помощью вызова стандартной функции PostgreSQL set_config :

```
perform set_config('LUI.RETVAL',ВозвращаемоеЗначение,false);
```

*Следует заметить, что “false” указывает системе PostgreSQL на необходимость сохранения значения параметра после успешного завершения текущей транзакции.

После выполнения программного кода вся конструкция динамического элемента заменится на значение параметра сеанса LUI.RETVAL.

Примеры:

```
{PLPGSQL:begin
perform set_config('LUI.RETVAL',version(),true);
exception
when others then
perform set_config('LUI.RETVAL','Error',true);
end;}
```

Возвращает версию СУБД

★ Question – Вопрос пользователю

{**Question**:***ТекстВопроса***;***Вариант1***:***Текст1***;…***ВариантN***:***ТекстN***;}

{**Q**:***ТекстВопроса***;***Вариант1***:***Текст1***;…***ВариантN***:***ТекстN***;}

При вычислении свойства с такой конструкцией вычислитель значения выражения приостанавливает свою работу, а клиенту показывают модальное окно с текстом вопроса, а также предлагают N вариантов ответа. В качестве ответов пользователь видит только Текст1 … ТекстN. Вариант1 … ВариантN – не отображаются.

Пользователь делает выбор варианта M и вся динамическая конструкция заменяется на ВариантM. После этого вычисление выражения продолжается. Если пользователь отказывается от выбора – вычисление прерывается, выполняется отказ от действия и делается попытка отката на момент начала действия, породившего этот вопрос.

Примеры:

- `SQL:{Question:Данные подготовлены, но ещё не сохранены. Необходимо принять решение;commit:Сохранить;rollback:Отменить;}`
  – в зависимости от выбора пользователя произойдёт или фиксация результатов работы в базе, или откат к началу транзакции.
- `{Question:Перестала ли ты пить коньяк по утрам?;Y:Да;N:Нет;:Не знаю;*:Может быть;}`

★ Calendar – Выбор даты/времени

{**Calendar**:[DATA_TYPE:***Тип***;][FORMAT:***Формат***;][DEFAULT:***НачальноеЗначение***;][FIELD:***Поле***;]}
```

Красные прямые скобки показывают необязательность каждой из 4-х частей конструкции:

- DATA_TYPE – тип данных. Тип может принимать 2 значения – DATE (только календарь) и DATETIME (Календарь и время). По умолчанию (если не указано) тип = DATE.
- FORMAT – это формат возвращаемого и начального значения. Если не указан, то формат DATE будет DD.MM.YYYY, а формат DATETIME будет DD.MM.YYYY HH24:MI:SS
- DEFAULT – начальное значение, которое сразу будет отображать календарь. Начальное значение должно соответствовать Формату.
- FIELD – код поля, около которого отображать календарь. Если не указать, то будет использоваться контекстное поле.

При вычислении свойства с такой конструкцией вычислитель выражения приостанавливает свою работу, а клиенту показывают модальное окно с календарём. Пользователь выбирает дату и время (если надо) и вся динамическая конструкция заменяется выбранным значением в указанном Формате. После этого вычисление продолжается. Если пользователь отказывается от выбора – вычисление прерывается, выполняется отказ от действия и делается попытка отката на момент начала действия, породившего этот вопрос.

Примеры:

- `{Calendar:FORMAT:DD.MM.YYYY;DEFAULT:13.02.2019;}` – заменится на выбранную дату в формате ДД.ММ.ГГГГ, причём сразу покажут дату 13 февраля 2019 года.
- `{Calendar:DATA_TYPE:{Property:DATA_TYPE};FORMAT:{Property:FORMAT};DEFAULT:{Property:Value};}` – заменится на выбранную дату в формате текущего поля, причём сразу покажут дату из текущего поля

★ Access – наличие права

{**Access**: ***ТипОбъекта***;***КодОбъекта***;***Право***}

Заменяется на Y, если указанное право на указанный объект указанного типа есть, или N, если права нет.

Примеры:

- `{Access:UI_ELEMENT;USERS;DEL}` – запрашивает доступ у текущего пользователя к действию «Удалить» в списке пользователей

★ HOST – результат выполнения команды ОС

{**HOST**: ***Код формы команды ОС***;***Код параметра1:Значение параметра1***;… ***Код параметраN:Значение параметраN***;}

Выполняет форму команды операционной системы (ОС), вычисляет закодированный в ней результат и возвращает вместо себя.

★ ENCR – шифрование строки

{**ENCR**:***Строка***}

Возвращает зашифрованную строку.

★ DECR – дешифрование строки

{**DECR**:***Строка***}

Возвращает расшифрованную строку.

★ UploadFile - загрузка файлов

{**UploadFile**:***Заголовок окна выбора файлов***}

Выполняет загрузку файлов с компьютера пользователя во временный каталог сеанса пользователя и возвращает перечень их имён разделённых точкой с запятой.

★ Parse – разбор строки

{**Parse**:***Строка***}

Языки динамических элементов могут возвращать конструкции, содержащие другие динамические элементы. Они уже не вычисляются – динамические конструкции становятся просто значениями.

Parse может принудительно вычислить такие повторные (рекурсивные) «замороженные» динамические конструкции! Это полезно, если свойства с динамическими элементами находятся вне метаданных, например, в специальных таблицах. Тогда динамикой такие данные вытаскиваются из таблиц, а командой Parse – довычисляются.

Примеры:

- `{Parse:{SQL:select my_property from ref_prop where code='{F:REF_CODE}'}}` – Вытаскивает строку из таблицы и вычисляет в ней динамические элементы
- `{Parse:{JS:"{"+("{F:F1}"||"{F:F2}")+"}"}}` – Значение поля F1 (а если оно пусто – F2) интерпретируется как строка с динамическими элементами. То же самое, в этом случае, можно было бы записать короче: `{{JS:"{F:F1}"||"{F:F2}"}}`

★ QuoteNull (или QN) – задвоение апострофов и null для пустой строки

{QuoteNull: ***Строка***}

Если строка пустая, то вернётся "null". Если строка не пустая, то апострофы будут задвоены и результат взят в одинарные апострофы. Удобно применять для перевода строки в вид пригодный для SQL.

Пример: {SQL:select {QN:{FormProperty:MyProperty}}}

★ QuoteEmpty (или QE) – задвоение апострофов

{QuoteEmpty: ***Строка***}

В переданной строке апострофы будут задвоены и результат взят в одинарные апострофы. Удобно применять для перевода строки в вид пригодный для SQL.

Пример: {SQL:select {QE:{FormProperty:MyProperty}}}

Команды процедурных языков

Формы LUI могут вызывать программы процедурных языков в следующих случаях:

- При активации действия формы (любым способом), перед, или вместо вызова другой формы, задачи или команды ОС (свойство действия PROC_INITIALLY).
- После завершения работы вызванной действием формы, задачи или команды ОС (свойство действия PROC_FINALLY)
- В момент загрузки очередной формы (свойство формы ON_LOAD)

Код процедурного языка предваряется префиксом с указанием языка:

КодЯзыка:*ЭлементыЯзыка*

Несколько программ могут последовательно следовать друг за другом. Программы отделяются минимум одной пустой строкой. Следствием этого является то, что внутри кода одной программы не должно быть пустых строк.

★ DBConnect – подключение к базе данных

DBConnect:***КодСоединения***;MODE:[COMMON|PERSONAL];USERNAME:***Имя***;PASSWORD:***Пароль***;URL:***url***;SCHEMA:***схема***;SQL:Оператор выполняемый сразу после подключения;

Открывает соединение базой данных. Если соединение с базой данных с таким кодом уже есть оно будет предварительно закрыто. Первая команда DBConnect, выполненная в ходе работы пользователя в прикладной системе, вызовет установку свойства Connection уровня прикладной системы {ApplicationProperty:Connection}. Все последующие команды DBConnect не вызывают изменений этого свойства. Если соеди-

нение будет разорвано командой DBDisconnect, и это соединение указано в свойстве {ApplicationProperty:Connection}, то это свойство станет равным пустой строке, и вновь будет заполнено при выполнении очередной команды DBConnect. Формы списков, бланков и меню также являются носителем собственного свойства с кодом Connection – {FormProperty:Connection}. При вызове формы она получает значение этого свойства из параметра CONNECTION$, переданного ей среди выходных параметров действия в вызывающей форме. Если такой параметр не передаётся, то форма получает значение свойства Connection из свойства уровня приложения {ApplicationProperty:Connection}. Форма может изменить унаследованное значение свойства Connection в следующих случаях:

- Явная команда FormProperty:Connection=КОД_СОЕДИНЕНИЯ
- При выполнении основного запроса в списке или бланке и в свойстве QUERY_TEXT указан код соединения в круглых скобках перед текстом запроса к БД

КодСоединения идентифицирует соединение с базой данных и используется в языках SQL, PLPGSQL, LOVQ и в основных запросах списков и бланков. При работе прикладной системы, сервер приложений поддерживает служебное соединение с БД для чтения метаданных из таблиц схемы LUI. Это соединение имеет код “LUI$”. Оно мультиплексируется и разделяется всеми работающими приложениями в рамках одного сервера приложений.

MODE управляет режимом транзакций: COMMON – «короткие» транзакции с autocommit и отсоединением от БД после каждого оператора. PERSONAL – «длинные» транзакции, длящиеся на протяжении обработки многих событий. Началом и концом транзакций управляет разработчик, вставляя commit в нужных ему действиях.

USERNAME – имя пользователя БД;

PASSWORD – пароль пользователя БД;

URL указывает url доступа к БД;

SCHEMA указывает схему БД, если объекты базы не будут снабжаться указаниями имени схемы;

SQL этот атрибут позволяет выполнить некий SQL-оператор или даже программу сразу после установления соединения. Например:

```
do $$
begin
perform set_config('tcp_keepalives_idle','600',false);
end;
$$;
commit;
```

устанавливает параметр сеанса tcp_keepalives_idle

★ DBDisconnect – отключение от базы данных

DBDisconnect:***КодСоединения***;

Отключает соединение, идентифицируемое указанным кодом соединения.

★ OpenSession – инициирование сеанса в прикладной системе

OpenSession:***Имя пользователя прикладной системы***;

Выполняет создание сеанса пользователя прикладной системы и регистрирует его в БД сервера приложений.

Пример: OpenSession:my_user

По этой команде закрывается текущий сеанс (если был открыт) и создаётся новый сеанс пользователя "my_user" в текущем приложении.В таблице lui_t_session создаётся новая запись. После создания сеанса:

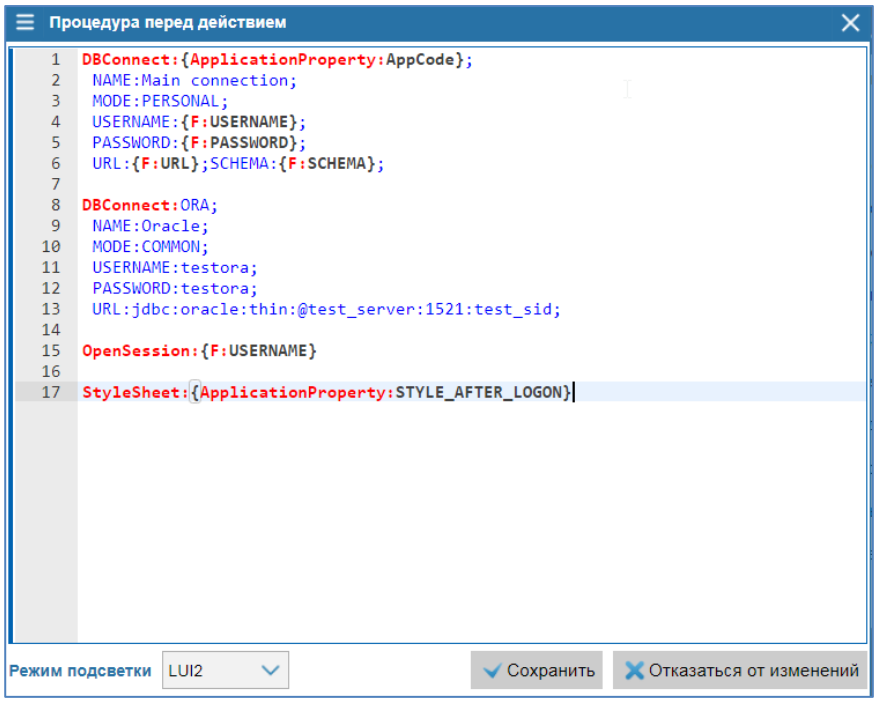
- станет возможным режим отладки и запись событий в lui_t_session_log

- можно обращаться к серверным функциям БД сервера приложений

nls, lui_r_show_message и многим другим.

Вызывать OpenSession следует в самом начале работы приложения. Например, по кнопке “OK” бланка LOGON.

Необходимость наличия этой команды в языке LUI обусловлена следующим. Во-первых, могут быть приложения, в которых нет необходимости запрашивать у пользователя логин и пароль. Во-вторых, только разработчик прикладной системы может точно определить момент, когда необходимо создавать сеанс, до подключения к БД или после подключения к БД. Пример действий по кнопке “OK” бланка LOGON:



В этой последовательности команд выполняется установка соединения с основной БД под именем и паролем, взятым из соответствующих полей бланка. Далее устанавливается соединение ещё с одной БД от имени пользователя testora. И только после успешно выполненных действий по подключению к этим БД создаётся сеанс в прикладной системе. После этого выполняется команда перехода на стиль оформления пользовательского интерфейса, который указан в параметрах приложения STYLE_AFTER_LOGON.

★ CloseSession – закрытие сеанса пользователя в прикладной системе

CloseSession:

По этой команде выполняется следующее.

Если сеанс пользователя находился в режиме отладки, то в таблице сеансов проставляется дата и время завершения сеанса. Если сеанс пользователя не находился в режиме отладки, то данные о сеансе удаляются из таблицы сеансов.

Затем выполняется закрытие всех соединений с базами данных инициированных приложением командами DBConnect.

Обычно эту команду применяют в действии EXIT с ролью CLOSE главного меню прикладной системы.

★ F – установка значения поля

F:***КодПоля*=*Значение***

Установка ***Значения*** в поле с кодом ***КодПоля***

★ ElementProperty – установка значения свойства элемента формы

ElementProperty:***КодЭлемента.КодСвойства*=*Значение***

Установка ***Значения*** свойства ***КодСвойства*** у элемента с кодом ***КодЭлемента*** в текущей форме

★ FormProperty – установка значения свойства формы

FormProperty:***КодСвойства***=***Значение***

Установка ***Значения*** свойства ***КодСвойства*** у текущей формы

★ ApplicationProperty – установка значения свойства Приложения

ApplicationProperty:***КодСвойства***=***Значение***

Установка ***Значения*** свойства ***КодСвойства*** у Приложения в текущем сеансе

★ JS – программа на JavaScript

JS:***ПрограммныйКод***;

Если в процессе выполнения программы на JavaScript будет создана глобальная переменная, совпадающая по имени с кодом некоего поля бланка:

- ***КодЭлемента*** = ***Значение***

то после окончания выполнения программы значение соответствующего поля бланка станет равным значению этой переменной.

В программе на JavaScript можно использовать конструкции типа:

- `Elements.`***КодЭлемента*****`.`*****Свойство*** = ***Значение***

Таким образом, можно создавать новые недостающие элементы (при загрузке формы), недостающие свойства новых или существующих элементов, а также менять существующие свойства нужных элементов.

Пример создания поля NAME в момент загрузки формы:

```
JS:Elements.NAME = {EType: "FIELD", CAPTION: "Имя", OPTIONAL: "N", Value: "Вася"};
```

В программе на JavaScript можно использовать конструкции типа:

- `Form.`***Свойство*** = ***Значение***

Таким образом, можно создавать или изменять значения свойств формы

В программе на JavaScript можно использовать конструкции типа:

- `Application.`***Свойство*** = ***Значение***

Так можно создавать или изменять значения свойств Приложения для конкретной сессии

★ PLPGSQL – программа на PL/pgSQL

PLPGSQL:(***КодСоединения***)***ПрограммныйКод***;

Код соединения в круглых скобках можно не указывать, если соединение с базой данных только одно.

Программа на PL/pgSQL может создавать новые элементы и модифицировать свойства элементов LUI посредством функции PL/pgSQL имеющейся в составе LUI.

- `Lui_r_api_set_element_property('`***КодЭлемента***`', '`***КодСвойства***`', '`***Значение***`');`

Если у текущей формы нет элемента с кодом ***КодЭлемента*** – он создастся. Если у элемента нет свойства ***КодСвойства***, то оно создастся. При этом указанному свойству указанного элемента будет присвоено ***Значение***.

Программа на PL/pgSQL может создавать и модифицировать свойства форм LUI посредством функции PL/pgSQL имеющейся в составе LUI.

- `Lui_r_api_set_form_property('`***КодСвойства***`', '`***Значение***`');`

Если у текущей формы нет свойства ***КодСвойства***, то оно создастся. При этом указанному свойству будет присвоено ***Значение***.

Программа на PL/pgSQL может создавать и модифицировать свойства приложения в текущем сеансе.

- `Lui_r_api_set_app_property('`***КодСвойства***`', '`***Значение***`');`

Если в текущем сеансе нет свойства приложения ***КодСвойства***, то оно создастся. При этом указанному свойству будет присвоено ***Значение***.

★ SQL – команда SQL

SQL:(***КодСоединения***)***КомандаSQL***

Выполнение операторов SQL. Если соединение только одно, то код соединения в круглых скобках можно не указывать. Любой возврат данных подавляется.

Примеры:

```
SQL:commit
```

★ HOST – команда операционной системы сервера приложений

HOST:***Команда***;***Параметр1***:***Значение1***;…***ПараметрN***:***ЗначениеN***;

★ HTML – генерация HTML-страницы

HTML:(Title:***Заголовок***;Target:***Значение***)***HTMLкод***

Отображает указанный HTML код

Target определяет способ отображения информации

- ***Значение*** = W – в отдельном окне браузера
- ***Значение*** = T – на отдельной вкладке браузера
- ***Значение*** = F – в плавающем псевдоокне
- ***Значение*** = M – в плавающем модальном псевдоокне

Пример

```
HTML:(Title:Sciprt for messages;Target:F;)<plaintext>{FormProperty:Script}
```

Вывод значения переменной Script текущей формы в псевдоокне с подавлением интерпретации html-тегов.

★ URL – открытие страницы на клиентском устройстве

URL: (Target:***Значение***;***Параметр1***:***Значение1;… ПараметрN***:***ЗначениеN***;)***URLадрес***

Target определяет способ отображения информации

- ***Значение*** = W – в отдельном окне браузера с минимальным набором управляющих элементов
- ***Значение*** = T – в новой вкладке браузера (по умолчанию)
- ***Значение*** = F – в плавающем псевдоокне

Заданные параметры передаются методом POST. Параметры, заданные в ***URLадрес*** передаются методом GET.

★ StyleSheet – установка стиля оформления пользовательского интерфейса

StyleSheet: ***КодСтиля***

По этой команде меняется внешний вид интерфейса пользователя в соответствии с указанным стилем.

Стили создаются инструментом “Редактор стилей”, входящим в поставку LUI.

★ File2Lob – загрузка файла в БД как большой бинарный объект

File2Lob: ***Имя файла;***SQL***:SQL-оператор; Remove:Y/N;***

File2Lob: ***Имя файла;***SQL***:(КодСоединения)SQL-оператор; Remove:Y/N;***

Имя файла может быть задано с полным путём или совсем без пути. Если путь не задан, то система будет считать, что файл создаётся во временном каталоге сеанса, имя которого хранится в свойстве приложения уровня сеанса {ApplicationProperty:TmpDir}.

Код соединения указывает, в каком соединении выполнять оператор SQL Если отсутствует используется соединение установленное в свойстве {FormProperty:Connection} текущей формы.

Оператор SQL - как правило, insert или update с одной единственной bind-переменной заданной в виде знака вопроса "?" Перед выполнением оператора SQL с этой bind-переменной будет связан поток чтения из указанного файла.

Remove:Y; - приведёт к удалению файла после успешной загрузки

Remove:N; - удаление выполняться не будет

Примеры:

File2Lob:T:\lui2\APP\usr\5567\image.png;SQL:(MY_CONN)insert into image_table(id,image)values(1,?);
File2Lob:image1.png;SQL:insert into image_table(id,image)values(1,?);Remove:Y;

★ Lob2File – выгрузка большого бинарного объекта из БД в файл

Lob2File:***SQL-оператор;***File: ***Имя файла;***

Lob2File: (***КодСоединения***)***SQL-оператор;*** File:***Имя файла;***

Имя файла может быть задано с полным путём или совсем без пути. Если путь не задан, то система будет считать, что файл создаётся во временном каталоге сеанса, имя которого хранится в свойстве приложения уровня сеанса {ApplicationProperty:TmpDir}.

Код соединения указывает, в каком соединении выполнять оператор SQL Если отсутствует используется соединение установленное в свойстве {FormProperty:Connection} текущей формы.

Оператор SQL – Запрос, извлекающий одну строку (только 1-я будет рассмотрена) и один столбец (только первый будет рассмотрен)

Пример:

Lob2File:select column::bytea from my_blob_table;File:C:\temp\my_blob_file.mp3;

★ unZipFile - разворачивание zip-архива в указанный каталог

unZipFile:***Имя файла;***To: ***Имя каталога;***

Если каталог не указан, то разворачивание выполняется в каталог {ApplicationProperty:TmpDir}.

Пример:

unZipFile:{ApplicationProperty:TmpDir}my_arch.zip;to:{ApplicationProperty:TmpDir}my_arch;

★ SendMail – отправка электронной почты

SendMail:***Тема***;Content:***Текст письма;*** ContentType:***Тип содержимого письма(html/plain)***;

Charset:***Кодировка текста письма(UTF-8/ windows-1251/…по умолчанию UTF-8)***;

Host:***SMTP-сервер (имя или IP)***;

Port:***Порт SMTP-сервера (по умолчанию25)***;

SSL:***Поддержка SSL (Y/N по умолчанию N)***;

TLS:***Поддержка TLS (Y/N по умолчанию N)***;

User:***Пользователь(Иногда SMTP-сервер требует именованного подключения)***;

Password:***Пароль пользователя***;

Sender:***Адрес отправителя***;

Recipients:***Адреса получателей (разделитель – запятая)***;

AttFiles:***Список полных имен файлов прикладываемых к письму(разделитель – запятая)***;

★ UploadFile - загрузка файлов

UploadFile:***Заголовок окна выбора файлов***

Выполняет загрузку файлов с компьютера пользователя во временный каталог сеанса пользователя.

★ ShowMessage – выдача сообщения

ShowMessage:***КодСообщения***;p1:***Значение1;***p2:***Значение2***;…

Отображает сообщение справочника сообщений. Сообщение идентифицируется по коду. При отображении конструкции типа <pN> заменяются соответствующими значениями из данной команды. Сообщения

вводятся и редактируются с помощью соответствующих списка и бланка в пункте меню “Настройки – Справочник сообщений”

★ RunApplication – запуск прикладной системы

RunApplication:*КодПриложения;*LANG:***КодЯзыка***;

Перезапуск прикладной системы. Текущее приложение будет закрыто и запущено указанное. Текущим языком будет указанный язык.

Такой вызов чаще всего применяют в начальной форме приложения для выбора языка при входе в прикладную систему.

Прикладные системы LUI

Прикладная система создаётся и изменяется в среде разработки прикладных систем. В главном меню среды разработки прикладных систем имеется пункт “Приложения”. По нему открывается список прикладных систем. Добавление прикладной системы выполняется действием “Добавить”. Удаление – действием “Удалить”. Изменение действием “Изменить”.

Прикладная система имеет следующие свойства, задаваемые при её создании:

Code – ***Уникальный в рамках экземпляра метаданных код***. Рекомендуется вводить краткое обозначение не более четырёх символов.

Name – ***Название прикладной системы.*** Если прикладная система должна поддерживать более одного национального языка, то название следует ввести на всех требуемых языках.

Description – ***Краткое описание прикладной системы.*** Если прикладная система должна поддерживать более одного национального языка, то название следует ввести на всех требуемых языках.

Notes – ***Примечание.*** Информация для разработчиков.

Помимо этих свойств, прикладная система может иметь множество свойств. Некоторые из них хранятся в БД в виде параметров настройки прикладной системы и в процессе работы прикладной системы создают соответствующие свойства в памяти сервера приложений. Хранимые свойства создаются и изменяются администратором прикладной системы в пункте меню “Настройки – Параметры” каждой прикладной системы.

Ниже приведён список параметров.

★ Параметры настройки

- **APPEARANCE** – Визуальное представление приложения. Возможны два значения:
  Windows – формы имеют оконный вид
  Trendy – формы не имеют традиционного оконного вида и отображаются в условно –модном стиле
- **DEBUG_MODE** – Режим отладки по умолчанию. Управляет начальным состоянием режима отладки после входа пользователя в приложение
- **FORM2RUN_SYNC_TASK** – Форма ввода параметров при запуске задачи в синхронном режиме. По умолчанию =LUI.LUI_B_TASK_RUN_SYNC. Т.е. форма с кодом LUI_B_TASK_RUN_SYNC из метаданных приложения LUI
- **FVCS_MODE** – Режим управления версиями форм. Возможные значения: N-управление версиями отсутствует; S-одна форма одновременно может модифицироваться только одним разработчиком;M- одна форма одновременно может модифицироваться несколькими разработчиками (разработчик формы может пригласить к разработке других разработчиков)
- **INITIAL_FORM** – Форма входа в приложение. Этот параметр указывает серверу приложений, какая форма должна быть выдана пользователю при входе. Как правило, это форма ввода имени и пароля.
- **MAIN_MENU_AUTOEXPAND** – Управляет необходимостью автоматического раскрытия пунктов (узлов) главного иерархического меню. Возможные значения: Y-главное меню представляется в виде, когда раскрыты основные узлы иерархии и отображаются пункты, которые вызывают формы прикладной системы; N – пункты меню свёрнуты в группы
- **MAIN_MENU_AUTORUNITEM** – Управляет необходимостью автоматического вызова форм прикладной системы при навигации по пунктам главного меню. Возможные значения: Y-форма вызывается при навигации клавиатурой или мышью на пункт меню предназначенный для её вызова. N – вызов формы выполняется нажатием Enter или двойным кликом мыши на пункте меню.

- **STYLE_BEFORE_LOGON** – Стиль оформления перед входом в приложение. Указывает код стиля для выдачи первой формы приложения. Стили создаются в редакторе стилей.
- **STYLE_AFTER_LOGON** – Стиль оформления после входа в приложение. Указывает код стиля для выдачи первой формы приложения.
- **TEMPLATE_CONFIGURATION** – Код конфигурации шаблонных форм. Устанавливает код конфигурации для всех шаблонных форм приложения, из которых берутся значения по умолчанию и шаблонные элементы прикладных форм. Как правило, = DEFAULT
- **VERIFIER_CONFIGURATION** – Код конфигурации форм верификаторов. Устанавливает код конфигурации для всех верификаторов приложения. Как правило, = DEFAULT
- **ABOUT_<КодЯзыка>** – Описание прикладной системы на языке html, которое вызывается из системного меню формы главного меню. Если приложение многоязычное, то рекомендуется иметь по одному параметру для каждого языка. Например: ABOUT_RUS и ABOUT_ENG.

Помимо свойств, хранящихся в БД в виде параметров настройки, прикладная система для каждого сеанса работы пользователя содержит в памяти сервера приложений следующие свойства:

★ Переменные сеанса

- **AppCode** – Уникальный код прикладной системы.
- **AppDirURL** – URL для доступа к файлам приложения. Каждой прикладной системе в файловой системе сервера приложений выделяется каталог, в котором прикладная система может хранить нужные ей файлы (скрипты, программы для интеграции и отчётов и т.п.) Это свойство устанавливает путь для браузера пользователя к этому каталогу.
- **AppFilePath** – Путь к файлам приложения в терминах операционной системы сервера приложений
- **AppName** – Название прикладной системы на текущем языке прикладной системы
- **AppUser** – Имя пользователя прикладной системы, который инициировал сеанс работы в прикладной системе
- **ApplicationURL** – URL введённый пользователем в браузере для доступа к прикладной системе
- **CURRENT_LANG** – Код текущего языка
- **ClientHostName** – IP-адрес или имя компьютера пользователя
- **Connection** – Код соединения с БД. Первая команда DBConnect, выполненная в ходе работы пользователя в прикладной системе вызовет установку свойства Connection уровня прикладной системы.
- **dbUser** – Имя пользователя соединения с БД, указанного в свойстве Connection
- **dbPassword** – Криптованный пароль соединения с БД, указанного в свойстве Connection
- **dbURL** – URL доступа соединения с БД, указанного в свойстве Connection
- **dbSCHEMA** – Схема данных соединения с БД, указанного в свойстве Connection
- **DEBUG_MODE** – Состояние режима отладки сеанса пользователя. В момент входа в приложение устанавливается путём чтения соответствующего параметра из БД.
- **PathSeparator** – Символ – разделитель имён каталогов в ОС сервера приложений
- **SessionID** – Уникальный идентификатор сеанса пользователя
- **TmpDir** – Путь к временным файлам сеанса приложения в терминах операционной системы сервера приложений
- **TmpDirURL** – Путь к временным файлам сеанса приложения в терминах браузера пользователя
- **Var FormID** – Идентификатор текущей активной формы

Все свойства и параметры прикладной системы доступны в метаязыке LUI через {**ApplicationProperty**:***КодСвойства***}

★ Вход в прикладную систему

Вход в прикладую систему, в том числе в среду разработки приложений выполняется с помощью браузера. В качестве URL можно ввести:

http://имяСервера:Порт/lui2/?AppCode=<Код прикладной системы>&Lang=<код языка>

Если не указан код языка, то текущим языком станет язык по умолчанию (***см. Поддержка национальных языков***)

Если не указан код прикладной системы, то будет осуществлён вход в среду разработки прикладных систем.

Структура URL зависит от конкретной установки LUI.

Параметры передаваемые в URL:

- **AppCode** – Код прикладной системы

Lang-Национальный язык зарегистрированный в прикладной системе, который будет языком “по-умолчанию” при входе в прикладную систему

- **Appearance**- Визуальное представление приложения.

Возможны два значения: Windows – формы имеют оконный вид,Trendy – формы не имеют традиционного оконного вида и отображаются в условно –модном стиле

- **DisableKeyDump** – Отключение формирования дампа памяти по клавише (обычно F5)

Формы LUI

Интерфейсные формы располагаются на окнах (или страницах). Все окна открываются в модальном режиме. То есть, фокус ввода находятся всегда в самой верхней форме. В одном окне может быть несколько форм разного типа, активных одновременно.

В рамках прикладной системы все формы идентифицируются уникальным алфавитно-цифровым кодом. Рекомендуется применять следующую мнемонику кодов форм

Префикс_Суффикс_Семантика

Префикс – краткое (желательно не более трёх символов) обозначение прикладной системы или её подсистемы (части)

Суффикс – L – для списков, B- для бланков, M- для меню, H – для команд ОС, T – для задач, V – для верификаторов, C – для диаграмм.

Семантика – краткое мнемоническое назначение формы.

Примеры.

LUI_L_USERS – список пользователей

CT_L_CONTRACTS – список договоров

CL_B_CLIENT – бланк редактирования данных о клиенте

GL_M_MAIN_MENU – главное меню

GL_H_PSQL – вызов команды ОС psql – CLI-интерфейс к PostgreSQL

Ниже следует подробное описание форм, их свойств и ожидаемого поведения.

Список

Основное предназначение Списков – поиск и отображение данных, полученных на основании SQL-***Запроса*** при выполнении которого заполняется таблица (***Grid***). Каждый ***Столбец*** Grid является элементом интерфейса со своими свойствами. Новые строки таблицы извлекаются (поступают в форму из источника выполнения SQL-запроса) по мере листания списка пользователем. Количество извлечённых строк не ограниченно. В режиме соединения PERCONAL (см. команду DBConnect – подключение к базе данных на стр. 15) соблюдается целостность чтения данных в списке в рамках одного запроса (данные соответствуют состоянию на момент начала выполнения запроса). Строки Grid не являются элементами интерфейса, но существует понятие контекстной строки. Текущим контекстом является набор значений ***Полей*** (***Ячеек***) текущей строки, которая явно выделяется строковым курсором.

Столбцы списка могут группироваться в ***Группы***, которые имеют некоторый тип: Static, Overflow или Virtual. Столбцы из ***Static*** групп располагаются вначале Grid и не прокручиваются по горизонтали. Столбцы из ***Overflow*** групп отображаются в виде отдельных полей в области справа от Grid. Любая группа может объединять столбцы общим заголовком. Не допускается вложенность групп друг в друга. Несколько групп Overflow образуют группу закладок справа от Grid.

Пользователь может в форме-списке перейти в режим ***QBE*** (Query by Example, см. стр.**Ошибка! Закладка не определена.**). В этом режиме для разных полей может быть доступен перечень возможных или типичных значений. Строки Grid, отображаемые после выполнения QBE, удовлетворяют заданным условиям. Поля в Overflow группах также могут участвовать в формировании QBE.

Под таблицей (Grid) может располагаться строка с ***Итоговыми полями***. Формула вычисления итогового значения работает над всеми строками в таблице (даже ещё не извлечёнными), с учётом отбора по условиям QBE.

Ячейки имеют свойство – «класс отображения» (VISUALIZATION). Разные классы умеют выделять ячейки цветом, шрифтом или другими стилями.

В том случае если для столбца указано свойство “Выражение для подсветки фрагментов” (MAKEUP) в ячейках может выделяться не только весь текст, но и его отдельные фрагменты.

Пользователь может перевести форму-список в режим ***многострочного выделения*** (явно или кликом мышкой с нажатым Ctrl). В этом режиме можно выделить несколько строк Grid, выделить все строки (даже ещё не извлечённые, в этом случае извлекаться строки будут уже в состоянии выделения), снять все выделения и инвертировать выделения строк. В режиме текущей строки она всегда является единственной выделенной строкой.

Пользователь может выполнять ***Действия*** в форме-списке. Действия могут относиться ко всему списку («Добавить», «Найти»…), к текущей строке («Редактировать», «Показать детали»…), к нескольким строкам («Удалить», «Обработать»…) или к текущей ячейке. Активизация действия может производиться различными способами: выбором пункта из локального иерархического меню, нажатием кнопки с пиктограммой на ToolBar или в ячейке, нажатием кнопки с текстом внизу формы... Кроме того, действия могут быть привязаны и выполняться автоматически при событиях: нажатие ролевой клавиши (Insert, Delete, Enter…), перемещении по строкам, закрытии формы, по таймеру и т.п.

Действия, работающие в отношении нескольких строк в режиме многострочного выделения, могут работать и в режиме текущей строки. Например, «Удалить».

Для быстрого выполнения действий можно использовать функциональные клавиши:

Клавиша	Действие	Описание
F1	Подсказка	Вызов справки о форме
F5	Дамп приложения	Получить информацию о элементах активной формы
F6	Добавить	Вызывается форма для добавления элемента списка
Shift+F6	Удалить	Удаляется текущий или выделенные элементы списка
F7	Ввод запроса	Включается режим ввода запроса для поиска элемента списка
F8	Выполнение запроса	Выполняется поиск элемента списка по условию запроса
F9	Список возможных значений	В режиме ввода запроса открывается список возможных значений

У любого действия есть признак ***доступности*** – свойство, как правило, имеющее динамическое значение, опирающееся на входные параметры и контекстные значения полей. При активизации пользователем конкретного действия (всё равно – каким способом) непосредственно перед его выполнением снова проводится контроль применимости действия (динамическое значение применимости перевычисляется).

Действия в форме-списке могут вызывать программу на процедурном языке и/или команду вызова другой формы. После выполнения Действия форма может перезапросить данные в текущей ячейке, во всей текущей строке или обновить Grid целиком (полный перезапрос). Действия могут иметь ***выходные параметры***, которые определяют значения ***входных параметров*** в вызываемой этим действием форме.

Действие в отношении ячейки может иметь специальное назначение – применение изменения текста в ячейке. В этом случае ячейка прямо в Grid является модифицируемой. При попытке уйти из ячейки выполняется действие, применяющее модифицированный текст.

Пользователь может менять ***порядок сортировки*** строк в Grid списках. Этот порядок вместе с условиями отбора в QBE может быть сохранён пользователем как персональная конфигурация списка под некоторым именем. Перечень сохранённых конфигураций всегда легкодоступен в списковой форме. Сохранённые пользовательские конфигурации могут стать доступными в основном меню, а одна из них может быть назначена как конфигурация по умолчанию и вызываться сразу при открытии формы.

Пользователи могут модифицировать ***интерфейс*** списков в разумных пределах: менять видимость, порядок и ширину столбцов, размер всего окна и пропорции областей внутри него. Эти изменения привязываются к текущей конфигурации (базовой или пользовательской) и действуют при последующих открытиях формы.

Форма-список может иметь ***входные параметры***. Ссылки на них используются в свойствах элементов формы, как правило – в динамических значениях. Входные параметры могут иметь значения по умолчанию, которые действую в случае, если форма вызывается без указания этого входного параметра. Значения по умолчанию могут быть динамическими и ссылаться на другие входные параметры.

★ Общие свойства Списка

Code – *Уникальный в рамках прикладной системы код формы*. Этот код вводится при создании списка и не меняется в разных его конфигурациях. По этому коду в совокупности с кодом конфигурации выполняется идентификация и загрузка формы списка из базы данных для выполнения в прикладной системе. Этот код указывается в действиях форм, которые вызывают список. Уникальный код может быть изменён в редакторе форм LUI. При этом, автоматических изменений кода в действиях вызывающих форм не выполняется. Это надо делать вручную. Рекомендуется, чтобы код списка состоял из:

- Краткого обозначения прикладной системы
- Суффикса “L”
- Мнемоники отражающей смысл команды

Пример: TST_L_DEPT – список отделов в прикладной системе TEST

QUERY_TEXT – ***Основной запрос Списка***. SQL-запрос, формирующий Grid Списка. Вместо столбцов запроса необходимо указывать *, обязательно обрамлённую пробелами! Не следует указывать в запросе фразу ORDER BY - для этого есть отдельные свойства столбцов списка. При выполнении этого запроса вместо звёздочки будут подставлены выражения, определяемые в каждом столбце Списка. В самом начале, перед Select, в круглых скобках может быть указан код соединения (если в Приложении используется более одного соединения с базами данных). Если код соединения не будет указан, то будет использоваться соединение, код которого к моменту выполнения запроса хранится в свойстве формы **Connection**. Это свойство формы наследуется из приложения или передаётся в параметре CONNECTION$ и может быть изменено самой формой. См. описание команды DBConnect – подключение к базе данных.

ON_LOAD – ***Процедура при загрузке списка***. Здесь могут размещаться команды процедурных языков, которые добавляют, удаляют или модифицируют свойства элементов Списка, загруженного из метаданных (см. раздел «Команды процедурных языков» на стр.15). Язык определяется префиксом. Программный код не нужно заключать в фигурные скобки, так как это не динамика, а, собственно, само значение свойства, что не исключает использование динамики для генерации всего кода или его части.

HEADER – ***Заголовок списка***. Если список создаёт своё окно, то данный текст отображается как заголовок окна. Если это не первая форма в окне, то такой текст отображается как подзаголовок. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

START_MODE – ***Начальный режим выделения строк***. Каждый Список может находиться в одном из двух режимов выделения строк: SINGLE_ROW – «Выделена только текущая строка», MULTI_ROW – «Многострочное выделение». Начальный режим выделения строк в процессе работы с формой может быть изменён пользователем. Любое другое значение будет открывать Список в режиме только текущей строки без возможности переключения в режим многострочного выделения. По умолчанию в новых Списках установлен именно последний режим.

NODE_TEXT – ***Текст узла дерева.*** Текст подписи к узлу дерева (при отображении списка в древовидной форме). Обычно здесь указываются ссылки на столбцы списка.

NODE_ICON – ***Иконка узла дерева.*** Иконка узла дерева (при отображении списка в древовидной форме). Обычно имя иконки генерируется из ссылок на столбцы списка.

NODE_STATE – ***Состояние узла дерева.*** Устанавливает состояние узла дерева при отображении списка в древовидной форме. Возможны следующие значения: “Leaf” - лист, “Collapsed” - свёрнут, “AutoExpand” - развёрнут. Если ни чего не указано, то поведение узла определяется сервером приложений самостоятельно (как правило будет “Collapsed”).

ADD_INFO – ***Информация о записи***. Текст, который будет помещён в информационное многострочное поле под списком и который будет меняться при навигации по строкам Grid. Таким образом, для каждой строки Списка может быть определён свой дополнительный текст.

FORM_NOTES – ***Пояснения в Help***. Текст описания текущей конфигурации Списка для отображения в Help и генерации документации.

FEATURE – ***Код привязки***. Это рабочий код Списка. Используется для ассоциации с ним пользовательских интерфейсных предпочтений, а также прав доступа к элементам для групп пользователей. По умолчанию код привязки совпадает с кодом метаданных, из которых был загружен Список

★ Свойство входных параметров Списка

В Списке могут быть описаны входные параметры с единственной целью – указать значение параметров по умолчанию, так как при динамической ссылке на неопределённый входной параметр её значение останется неизвестным. Для ссылки на значение входного параметра укажите `{Param:КОД}` или `{F:КОД}`, где КОД – это код специального элемента Списка типа PARAM - «Входной параметр». Следует заметить, что в отличие от столбцов параметры всегда неявно обладают типом данных CHAR. Поэтому, при кодировании свойств форм запись '**{F:КОД}**' всегда эквивалентна **{F2SQL:КОД}**.

Code – ***Код входного параметра.*** Уникальный код элемента среди прочих элементов формы. Задаётся при создании параметров. Код может быть изменён действием "Изменить" в навигаторе редактора форм. Именно по этому коду производится связывание выходных параметров действия в вызывающей форме с входными параметрами списка. Код параметра рекомендуется снабжать префиксом "P_". Пример: P_CODE.

Name – ***Название входного параметра.*** Задаётся при создании параметров. Используется для отображения в навигаторе редактора форм. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

VALUE – ***Значение по умолчанию***. Если вызывающая Список форма не укажет значение параметра, то при ссылке на данный входной параметр в динамических свойствах будет подставляться указанное тут значение. По умолчанию – пустая строка. Значение по умолчанию может содержать динамические элементы. Вся динамика в значениях по умолчанию вычисляется в момент загрузки списка, а также в случае переинициализации формы-списка после выполнения действия (см. Свойства действий Списка, свойство AFTER, значение REFRESH на стр.32)

Если вызывающая форма переопределила значение по умолчанию входного параметра, то указываемая здесь динамика вычисляться никогда не будет.

DESCRIPTION – ***Описание***. Комментарий разработчика о входном параметре и о том, на что он влияет.

★ Неявные и системные входные параметры Списка

CONFIG – ***Код конфигурации списка.*** Если ни в выходных параметрах действия, которое вызывает список, ни во входных параметрах самого списка не указан параметр CONFIG, то считается что он = DEFAULT. Т.е. будет загружаться конфигурация "по умолчанию" – DEFAULT.

POSITION$ – ***положение окна.*** Этот параметр управляет положением окна при его открытии. Возможные значения:

- Cascade – каскадом относительно родительского окна (по умолчанию);
- TopLeft – слева вверху экрана;
- WorkCenter-окно располагается в центре экрана;
- ParentRight-по-возможности вплотную справа от окна вызывающей формы;

WINWIDTH$ – ***ширина окна в пикселях.*** Этот параметр устанавливает ширину окна. Обычно сервер визуализации сам удачно подбирает ширину окна. Этот параметр позволяет разработчику отменить этот подбор. Этот параметр будет действовать, если ранее текущий пользователь ещё не изменял ширину окна данного списка, исходя из своих собственных предпочтений.

WINHEIGHT$ - ***высота окна в пикселях.*** Этот параметр устанавливает высоту окна. Обычно сервер визуализации сам удачно подбирает высоту окна. Этот параметр позволяет разработчику отменить этот подбор. Этот параметр будет действовать, если ранее текущий пользователь ещё не изменял высоту окна данного списка, исходя из своих собственных предпочтений.

CONECTION$ - ***код соединения с БД.*** Этот параметр следует применять установки кода соединения с БД и установки свойства Connection уровня формы. Если этот параметр не передан, то свойство Connection уровня формы будет унаследовано из свойства Connection уровня приложения. Значение свойства Connection может быть изменено в самом списке, например в свойстве ON_LOAD применена команда

FormProperty:Connection=КОД_СОЕДИНЕНИЯ, или в свойстве QUERY_TEXT перед тестом запроса указа код соединения в круглых скобках. Значение свойства Connection используется при выполнении команд **SQL:Команда**, вычислении динамических элементов **{SQL:Выражение}**, при выполнении основного запроса списка, если не указан код соединения в круглых скобках.

★ Свойства столбцов Списка

В Списке должны быть определены столбцы. Как минимум – один столбец. Каждое событие в Списке обрабатывается в контексте текущей строки (если строки вообще есть). Совокупность значений ячеек текущей строки и есть контекст. Динамические свойства элементов Списка могут ссылаться на значение ячейки столбца из контекстной строки. Для этого укажите **{F:КОД}** или **{F2SQL:КОД}**, где КОД – это код элемента типа «Столбец». Если контекст не определён, то все ссылки на столбцы Списка возвращают пустую строку.

Code – ***Код столбца.*** Уникальный код элемента среди прочих элементов формы. Задаётся при создании параметров. Код может быть изменён действием “Изменить” в навигаторе редактора форм.

DATA_TYPE – ***Тип данных***. Это свойство задаётся при создании столбца и имеет постоянное значение во всех конфигурациях списка. Может быть изменено действием “Изменить” в навигаторе редактора форм.

Указанный тип используется в процедуре конвертации полученных значений при выполнении SQL-запроса из свойства QUERY_TEXT. Тип данных используется в операторе языка LUI **{F2SQL:Код столбца}** для формирования конструкции пригодной для применения в языке SQL. Выражение **{F: Код столбца}** принимает текстуальное значение из соответствующей ячейки, независимо от типа данных.

Тип данных влияет на правильность сортировки и учитывается в запросах QBE (см. стр. **Ошибка! Закладка не определена.**).

Архитектор прикладной системы может добавлять новые типы данных и изменять свойства существующих типов данных. Поставляемые в составе LUI типы данных:

- BOOLEAN – логический тип данных. Возможные значения: Y-истина;N-ложь, NULL – пусто;
- CHAR – Текстовая строка (значение по умолчанию);
- CIDR – IP-сеть (на данный момент поддерживается только стандарт v4)
- DATETIME – Дата и/или время. Настоятельно рекомендуется использование формата данных (свойство FORMAT);
- DTRANGE – диапазон даты/времени. Составной тип, имеющий четыре части
    - Признак вхождения в диапазон нижней границы
    - Нижняя граница – дата/время
    - Верхняя граница – дата/время
    - Признак вхождения в диапазон верхней границы

  Существует специальное значение “empty” - пустой диапазон (не путать с NULL!) отображаемое как “()”.
  Нижняя и верхняя границы могут отсутствовать. Это означает бесконечность снизу или сверху.
  Настоятельно рекомендуется использование формата данных (свойство FORMAT). Формат применяется при извлечении данных из БД к каждой границе.
- INET - IP-адрес (на данный момент поддерживается только стандарт v4)
- INTERVAL – продолжительность во времени. Настоятельно рекомендуется использование формата данных (свойство FORMAT);
- MACADDR – MAC-адрес 6-ти байтная версия;
- MACADDR8 – MAC-адрес 8-ми байтная версия;
- NUMBER – Число. Возможно использование формата данных (свойство FORMAT);
- NUMRANGE – диапазон чисел. Составной тип, имеющий четыре части
    - Признак вхождения в диапазон нижней границы
    - Нижняя граница – число
    - Верхняя граница – число
    - Признак вхождения в диапазон верхней границы

  Существует специальное значение “empty” - пустой диапазон (не путать с NULL!) отображаемое как “()”.
  Нижняя и верхняя границы могут отсутствовать. Это означает бесконечность снизу или сверху.
  Возможно использование формата данных (свойство FORMAT). Формат применяется при извлечении данных из БД к каждой границе.
- NLSTEXT – Многоязычный текст. Список отображает значения на текущем национальном языке прикладной системы;
- RICHTEXT – Форматированный текст. В поле можно будет вставлять мультимедиа элементы, а также форматировать текст посредством текстового редактора;

ITEM_TEMPLATE – ***Шаблон элемента***. Это свойство задаётся при создании столбца и имеет постоянное значение во всех конфигурациях списка. Обычно оно автоматически заполняется редактором форм, если для указанного типа данных в шаблонном списке LIST_TEMPLATE имеется столбец с кодом **ITEM_тип данных**. Шаблон определяет свойства столбца "по умолчанию". Значение может быть изменено действием "Изменить" в навигаторе редактора форм.

Seq – ***Порядковый номер элемента.*** Это свойство задаётся при создании столбца и имеет постоянное значение во всех конфигурациях списка. Это свойство задаёт положение столбца в окне списка относительно других элементов расположенных на том же уровне (т.е. в той же группе или на уровне формы в целом). Порядковый номер можно изменять действием "Изменить" в навигаторе редактора форм. Или действиями "Вверх/Вниз" в бланке свойств элемента редактора форм.

Name – ***Название элемента.*** Это свойство задаётся при создании столбца и имеет постоянное значение во всех конфигурациях списка. Может быть изменено действием "Изменить" в навигаторе редактора форм. Название используется для отображения в навигаторе редактора форм и для документации.

Parent – ***Родительская группа.*** Это свойство задаётся при создании столбца и имеет постоянное значение во всех конфигурациях списка. Может быть изменено действием "Изменить" в навигаторе редактора форм. Столбец в списке располагается либо в прокручиваемой горизонтально группе таблицы ***(Grid)*** - уровень формы, либо в одной из групп столбцов.

DEFINITION – ***SQL–выражение***. Выражение, которое будет добавлено в основной SQL-запрос (вместо звёздочки) через запятую, наряду с SQL-выражениями других столбцов. Значение этого выражения даёт значение ячейки для каждой строки Grid.

FORMAT – ***Формат данных***. Указание формата обязательно для полей с типом данных DATETIME и возможно для полей с типом данных NUMBER.

FOR_UNIQUE_KEY – ***Входит в уникальный ключ строки***? В любом Списке должен быть определён набор столбцов, составляющий уникальную комбинацию данных, однозначно определяющую каждую строку списка. Значения, порождённые из столбцов, входящих в уникальный ключ, используются при перезапросах (одной записи или всего списка). Возможные значения:

- Y – Столбец входит в уникальный ключ;
- N – Столбец не входит в уникальный ключ;
- Таблица.Столбец; (Пример: MY_TABLE.ID) -столбец является частью первичного ключа и явно отображён на столбец таблицы БД

По умолчанию – значение N.

Указание явного отображения позволяет избежать некоторых коллизий при генерации запросов блокировки строки в случае, когда данные в списке извлекаются из многих таблиц сложным запросом.

Это свойство используется сервером приложений:

- для генерации запроса блокировки строки, если действие, выполняемое в списке, требует блокировки;
- для организации перезапроса данных строки, если действие, выполняемое в списке, требует обновления данных в строке;
- для идентификации строк списка при выполнении действий

Настоятельно рекомендуется в каждом списке иметь хотя-бы один столбец входящий в уникальный ключ.

CAPTION – ***Заголовок столбца***. Текст над колонкой или полем (если столбец в области Overflow). Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

ALIGNMENT – ***Выравнивание***. Выравнивание текста внутри ячейки:

- LEFT – По левому краю (по умолчанию);
- CENTER – По центру;
- RIGHT – По правому краю.

Любое другое значение (включая пустую строку) эквивалентно значению LEFT.

VISIBLE – ***Доступен***? Признак доступности столбца пользователю:

- Y – столбец доступен (по умолчанию). Даже если он изначально не видим, пользователь может сделать его видимым;
- Любое другое значение (включая пустую строку) – столбец недоступен и пользователь не видит его при настройке отображения, и не может сделать видимым.

WIDTH – ***Признак видимости столбца***. Отображать ли столбец при первом открытии Списка. Значение 0 (ноль) говорит о невидимости столбца. Любое другое значение (включая пустую строку) означает видимость столбца. Пользователь в процессе работы с формой может изменять видимость столбцов, а также устанавливать им ширину. По умолчанию – значение 0.

VISUALIZATION – ***Код набора свойств отображения***. Способ визуально выделить ячейку с данными. По сути это – имя css-класса для стилевого оформления ячейки. Автоматически поддерживаются следующие варианты:

- FONT_SELECTED – Выделен текст;
- BACKGROUND_SELECTED – Выделен фон;
- ALL_SELECTED – Выделен текст и фон;
- LOWERED – слабозаметный;
- DANGER – Опасность!;
- ATTENTION – Внимание!;
- OK – Успех;
- INHERIT – Унаследованое значение;
- SECTION – Подраздел;
- START – Включить/Работает;
- INTERIM – Приостановить/Переключается;
- STOP – Остановить/Выключено.
- NOTES – Пояснения, дополнительные сведения

Можно добавлять свои собственные классы визуализации.

MAKEUP – ***Выражение для подсветки фрагментов текста в ячейках***. Здесь можно ввести одно или более регулярных выражений для применения стилей визуализации к подстрокам соответствующим этим выражениям. Пример:

если в это поле введено /(d{4})\.(d{4})\.(d{2})\.(d{2}) и в свойстве "Класс отображения" указано NORMAL DANGER INHERIT OK, и поле содержит

2019.07.01, то всё поле будет отображаться классом NORMAL, год - классом DANGER, месяц - классом INHERIT, день - классом OK

HINT – ***Подсказка***. Текст, отображающийся в качестве пояснения к столбцу или значению ячейки столбца в конкретной записи. Рекомендуется вместить пояснения в одну ёмкую, но предельно точную фразу. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

SORT – ***Порядок сортировки***. Число, отражающее порядок столбца в общей сортировке данных в Grid. Пустая строка – поле не участвует в сортировке. Отрицательное число – обратная сортировка. Лидирующий ноль – пустые значения в начале.

VARIANTS – ***Варианты для QBE***. Предлагает пользователю список типичных значений для данного поля в режиме задания условия Query by Example (см. стр.**Ошибка! Закладка не определена.**). Здесь можно указать:

- SQL - запрос. Будут отображаться только столбцы с алиасами в двойных кавычках. Выбранный результат – значение из первого столбца;
- Коллекцию вариантов в формате Значение1:Пояснение;Значение2:Пояснение;... Выбранный результат – значение выбранного варианта

Запрос и коллекцию не нужно заключать в фигурные скобки, ибо это не динамика, а, собственно, значение свойства, что не исключает использование динамики для генерации запроса или коллекции

Первый вариант отображается в виде модального элемента выбора. Второй вариант отображается в виде выпадающего списка у ячейки списка.

Система сама может разобраться, какой вариант подразумевал разработчик. Но в сложных случаях рекомендуется делать подсказки в виде префиксов LOVQ: и LOVD:.

TOTAL – ***Итоговое значение***. Значение, отображающееся в итоговой строке списка, идущей сразу за Grid вне вертикальной прокрутки. Здесь надо указать групповую SQL-функцию, например SUM(). Аргументами могут выступать коды столбцов списка. Итоговая строка отображается, только если есть хотя бы один столбец с непустым TOTAL. По умолчанию автоматическое вычисление значений итоговой строки не гарантируются. Необходимо сделать двойной клик мышью в любой ячейке итоговой строки. Для гарантии автоматического вычисления значения итоговой строки добавьте в основной запрос (свойство DEFINITION) в любое место текст `/* CALCULATE_TOTAL_IMMEDIATE */`

ITEM_NOTES – ***Пояснения в Help***. Текст описания столбца Списка для отображения в Help и генерации документации. Следует помещать сюда всё, что не уместилось в одну фразу в свойстве HINT. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

★ Группировка столбцов Списка

Столбцы списка могут объединяться в группы с различными целями. Самая простая цель – указать для группы столбцов общий заголовок. Пользователь в процессе использования Списка может самостоятельно перемещать столбцы из группы в группу. Эти перемещения запоминаются и автоматически привязываются к пользовательской конфигурации.

Свойства групп Списка:

Code – ***Код группы.*** Уникальный код элемента среди прочих элементов формы. Задаётся при создании параметров. Код может быть изменён действием “Изменить” в навигаторе редактора форм.

Seq – ***Порядковый номер группы.*** Задаётся при создании. Это свойство задаёт положение группы в окне списка относительно других элементов расположенных на том же уровне (т.е. в той же группе или на уровне формы в целом). Порядковый номер можно изменять действием “Изменить” в навигаторе редактора форм. Или действиями “Вверх/Вниз” в бланке свойств элемента редактора форм.

Name – ***Название группы.*** Задаётся при создании. Используется для отображения в навигаторе редактора форм. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

GROUP_TYPE – ***Тип группировки***. Возможны варианты:

- FIXED – Столбцы внутри группы не прокручиваются по горизонтали – всегда видны в начале Grid. У группы возможен общий заголовок.
- OVERFLOW – Столбцы внутри группы отображаются как поля вне Grid права от него. При наличии нескольких групп типа OVERFLOW, они образуют блок закладок.
- Любое другое значение, включая пустую строку – виртуальная группа обычных столбцов, создаваемая исключительно ради отображения общего заголовка.

Значение по умолчанию – пустая строка.

TITLE – ***Общий заголовок***. Заголовок, объединяющий входящие в группу столбцы. Для групп типа OVERFLOW – это надпись на закладке в области переполнения (справа от Grid). Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

POSITION – ***Положение подписей***. Расположение подписей к полям внутри группы Overflow:

- AUTO (по умолчанию) – решает движок, как считает более подходящим. В настоящий момент – всегда сверху от полей;
- LEFT – слева от всех полей;
- UP – сверху от всех полей.

Любое другое значение эквивалентно значению UP.

GROUP_NOTES – ***Пояснения в Help***. Текст описания группы столбцов для отображения в Help и генерации документации. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

★ Свойства действий Списка

Действия, каким бы способом они ни активировались, выполняются в контексте текущей формы. Если только этим и ограничивается, то действие производится в отношении всего Списка (добавить новый элемент списка, изменить условия отображения всего списка, удалить, подтвердить обработать все строки и т.п.). Действия могут выполняться в контексте текущей строки (удалить строку, изменить строку, показать

детали строки и т.п.). Контекст может быть расширен на несколько (выделенных) строк. Действия могут быть локализованы внутри одной ячейки некоторой строки.

Code – ***Код действия.*** Уникальный код элемента среди прочих элементов формы. Задаётся при создании параметров. Код может быть изменён действием “Изменить” в навигаторе редактора форм.

Seq – ***Порядковый номер действия.*** Задаётся при создании. Это свойство задаёт положение пункта меню или иконки в панели инструментов относительно других элементов расположенных на том же уровне. Порядковый номер можно изменять действием “Изменить” в навигаторе редактора форм. Или действиями “Вверх/Вниз” в бланке свойств элемента редактора форм.

Name – ***Название действия.*** Задаётся при создании. Используется для отображения в навигаторе редактора форм. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

PROMPT – ***Наименование действия в интерфейсе пользователя***. Заголовок действия в локальном меню, отображаемом по нажатию правой кнопки мыши. Если PROMPT пуст, то соответствующего пункта локального меню не будет. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

В тексте можно использовать вертикальную черту. Тогда текст до вертикальной черты будет образовывать узел в локальном меню, а текст после черты – текст пункта подменю, раскрываемом подузлом. Глубина вложенности иерархии меню не ограничена.

В любое место текста можно вставить конструкцию `[ico:`***Имя***`]` где «Имя» - название иконки. В этом случае прямо в данном месте текста при выводе локального меню будет отображаться указанная иконка. Чтобы избежать копирования со свойством ICON, следует использовать {Property:ICON}. Пример:

```
[ico:{Property:ICON}]Добавить
```

EXPLANATION – ***Подсказка***. Текст, отображающийся в качестве пояснения к действию. Рекомендуется вместить пояснения в одну ёмкую, но предельно точную фразу. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

ACTIVE – ***Признак активности***. Это свойство проверяется как минимум 2 раза. Первый – при отображении элемента управления, активирующего данное действие (активно/не активно), второй – непосредственно перед выполнением Возможные значения:

- Y – Действие доступно для выполнения любым возможным способом активации (по умолчанию);
- EXIST – Действие применимо только к текущей записи (записям). Не применимо к списку в целом;
- Любое другое значение (включая пустую строку) – Действие не доступно для выполнения, каким бы способом оно ни активировалось;

DISPLAY – ***Признак отображения в локальном меню***. Можно ли активировать действие из локального меню, отображаемого по нажатию правой кнопки мыши. Если действие не активно, но отображается в меню, то текст пункта меню будет серым цветом и вызвать его будет нельзя. Возможные значения:

- Y - Действие отображается в локальном меню (при наличии непустого PROMPT);
- EXIST – Действие отображается только если существует текущая запись (или есть выделенные записи);
- Любое другое значение (включая пустую строку) – Действие не отображается в локальном меню.

Значение по умолчанию синхронизировано со свойством ACTIVE (ссылается на значение этого свойства: `{Property:ACTIVE}`). Действия из одного десятка по порядку, отделяются в локальном меню от других действий сепаратором. Действия ячеек всегда отображаются вначале меню.

В качестве значения тут можно указать запрос (без фигурных скобок), в котором алиасы столбцов должны совпадать с кодами свойства действия. Для каждой строки, порождаемой запросом, будет образовываться отдельный пункт меню (следствие: столбец PROMPT обязательно должно быть в запросе). Выбор некоторого пункта меню, порождённого этим запросом, выполняет это действие с переопределёнными запросом свойствами.

ICON – ***Иконка***. Другой способ активации действия – нажатие на иконку. Для глобальных действий иконка отображается в виде кнопочки на Tool-Bar над Grid, для действий в ячейках иконка отображается

прямо в ячейке. Если данное свойство пусто, то никакая иконка не отображается и место для действия для неё не резервируется. Если есть нескольких действий с одинаковой иконкой, то эта иконка отображается только в одном экземпляре. При клике на такой иконке выполняется первое по порядку активное действие с данной иконкой.

BUTTON – ***Текст на кнопке***. При непустом значении этого свойства внизу формы отображается кнопка с этим текстом, активирующая действие. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

ROLE – ***Способы активации***. Альтернативные способы активации действия:

- DBLCLICK – Двойной клик левой кнопкой мыши или нажатие клавиши `Enter`;
- ADD – Нажатие клавиши `Insert`. По смыслу – действие добавляет строки в список;
- DELETE – Нажатие клавиши `Delete`. По смыслу – действие удаляет строки из списка;
- CLOSE – Нажатие клавиши `Esc` или закрытие окна. Выполняется также при попытке закрытия формы, каким бы способом это закрытие не выполнялось;
- AUTO – Действие, выполняющееся при каждом изменении значения любого его выходного параметра;
- TIME – Действие, выполняющееся ежесекундно по таймеру (в периоды активности формы);
- WIZARD – Действие, являющееся очередным шагом визарда (вперёд или назад);
- POSTEDIT – Действие в ячейке, выполняющееся после изменения данных в ней.

Каждый способ активации выполняет первое по порядку активное действие, которому назначен данный способ активации. Способы активации можно комбинировать, указывая их через пробел и/или запятую.

APPLY_TO – ***Применимо к***… Указание, каким образом будет производиться обработка строк при выполнении действия:

- N – Действие будет применяться только к одной записи – текущей. Или же ни к какой строке, а к списку в целом. В действии возможны ссылки на текущие значения любых столбцов. В режиме многострочного выделения такое действие будет не активным;
- Y – Действие будет применяется к нескольким выделенным строкам. Ссылки на текущие значения столбцов бессмысленны. Обращение к {FormProperty:CheckedRows} вернёт перечень выделенных строк (а в режиме инверсии выделения – развыделенных строк) в виде строки в формате JSON. В режиме однострочного выделения такое действие будет не активным;
- PROGRESS – Действие будет применяться к текущей (в режиме однострочного выделения) или к выбранным (в режиме многострочного выделения) записям. В действии возможны ссылки только на столбцы из уникального ключа. Процедуры из PROC_INITIALY и PROC_FINALY будут выполняться для каждой выделенной строки, если их код отличается для разных выделенных строк;

Любое другое значение (включая пустую строку) эквивалентно значению N. По умолчанию свойство имеет значение N.

QUESTION – ***Текст предупреждения***. Если свойство не пусто, то его значение будет выдано как предупреждения с двумя кнопками. Если текст заканчивается знаком вопроса, то на кнопках будут надписи «Да» и «Нет». Иначе на кнопках отобразится «Продолжить» и «Отменить». Выбор первой кнопки позволяет продолжить выполнение действия. Выбор второй кнопки – приводит к прекращению выполнения действия и откату к началу его выполнения.

LOCK – ***Накладывать ли блокировку***. Перед выполнением действия в отношении текущей строки (APPLY_TO = N, ACTIVE = EXIST) может быть произведена попытка наложения на неё блокировки посредством выполнения основного запроса списка с дополнительными условиями на значения столбцов, входящих в уникальный ключ списка. Блокировка накладывается на таблицу, содержащую столбцы, входящие в уникальный ключ списка (FOR_UNIQUE_KEY = Y). При неудаче производится попытка наложения блокировки на записи всех таблиц из основного запроса. Если и в этом случае наложить блокировку не получилось, то блокировка не накладывается без выдачи сообщения об ошибке. Варианты:

- Y – выполнять попытку блокирования текущей записи (по умолчанию);
- N – не выполнять блокировку;
- Собственный оператор SELECT FOR UPDATE;

Использование собственного оператора отменяет генерацию оператора сервером приложений.

Пример:

```
select * from my_table where id={F:ID} for update of my_table
```

PROC_INITIALLY – ***Программный код действия***. Здесь указывается текст на процедурном языке, выполняющийся при активации действия перед (или вместо) вызовом формы, команды OS и т.п. Язык определяется префиксом (см. «Команды процедурных языков» на стр.15). Программный код не нужно заключать в фигурные скобки, так как это не динамика, а, собственно, само значение свойства, что не исключает использование динамики для генерации всего кода или его части.

Можно последовательно указать несколько программ на разных процедурных языках. В этом случае языки должны отделяться друг от друга как минимум одной пустой строкой. Следствие: внутри кода процедурного языка не должно быть пустых строк.

В случае если свойство APPLY_TO = PROGRESS, данный программный код (коды) выполняется для каждой выделенной строки.

COMMAND_TYPE – ***Тип вызываемого действия***. Возможны следующие значения:

- LIST – Вызов формы LUI. В этом случае свойство COMMAND – код списка или бланка из метаданных;
- BLANC – Вызов формы LUI. В этом случае свойство COMMAND – код списка или бланка из метаданных;
- TREE - Вызов иерархической структуры LUI. В этом случае свойство COMMAND – код списка из метаданных, который образует элементы первого уровня иерархической структуры;
- HOST – Вызов внешней команды OS на сервере приложений. В этом случае свойство COMMAND – команда OS из списка явно разрешённых команд;
- URL – Вызов WEB-страницы на клиентском устройстве. В этом случае свойство COMMAND – URL страницы;
- MENU –Вызов формы LUI. В этом случае свойство COMMAND – код формы LUI типа «Меню»;
- TASK – Вызов бланка запуска Задачи LUI. В этом случае свойство COMMAND – код Задачи, запускающейся в синхронном режиме.

COMMAND – ***Объект действия***. Вызываемый объект, соответствующий типу действия, указанному в свойстве COMMAND_TYPE. Помимо традиционного способа передачи параметров (отдельными подчинёнными элементами действия типа PARAM), возможно перечисление параметров в коллекции, указанной в круглых скобках сразу после объекта - идентификатора действия:

```
Объект(Параметр1:Значение;Параметр2:Значение;...;).
```

Даже в случае действия над несколькими отмеченными строками, вызов указанного тут объекта выполняется только 1 раз.

PROC_FINALLY – ***Программный код после действия***. Здесь указывается текст на процедурном языке, выполняющийся после вызова и завершения формы, задачи, команды OS и т.п. Язык определяется префиксом. Программный код не нужно заключать в фигурные скобки, так как это не динамика, а, собственно, само значение свойства, что не исключает использование динамики для генерации всего кода или его части.

Можно последовательно указать несколько программ на разных процедурных языках. В этом случае языки должны отделяться друг от друга как минимум одной пустой строкой.

В случае если свойство APPLY_TO = PROGRESS, данный программный код (коды) выполняется для каждой выделенной строки.

Если вызываемая командой COMMAND форма (Список, Бланк или Иерархическая структура) закрылась в режиме CANCEL (действие в вызываемой форме, выполняющееся по закрытию, имеет свойство AFTER = CANCEL), то код из PROC_FINALLY не выполняется, действие отменяется, а транзакция откатывается к началу действия.

CHILD_COMMIT –***Разрешать завершать транзакцию в вызываемой форме?*** В настоящий момент не действует.

AFTER – ***Поведение Списка после выполнения действия***. После успешного выполнения действия в случае, если вызывающая форма не закрылась в режиме CANCEL, текущий список поведёт себя в соответствии с указанными тут вариантами:

- NONE – Ничего не произойдёт, даже выделенные строки не сбросятся;
- FIELD – Обновится значение текущего поля текущей строки Списка (в котором стоит курсор);
- CURRENT – Обновится вся текущая строка Списка;

- ALL – Обновятся все строки Списка, то есть сделается перезапрос Grid. Текущая строка попытается сохраниться, но это не гарантируется. Все выделенные строки сбросятся;
- REFRESH – Переинициировать Список, то есть, как бы выйти и войти с теми же значениями входных параметров. Код из свойства ON_LOAD выполнится повторно;
- EXIT – Закрыть Список и передать управление в вызывающую форму. В случае визарда закроется весь визард;
- CANCEL – Отменить вызов Списка. В вызывающей форме не будут выполняться POST-действия и перезапросы. В случае визарда откатится весь визард. Для отката только на предыдущий шаг визарда данное действие должно иметь ROLE = WIZARD.
- QBE – переход в режим QBE

Любое другое значение, включая пустую строку, эквивалентно NONE.

ACTION_NOTES – ***Пояснения в Help***. Текст описания действия Списка для отображения в Help и генерации документации. Следует помещать сюда всё, что не уместилось в одну фразу в свойстве EXPLANATION. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

★ Свойства выходных параметров действий Списка

Подчинённые действию элементы типа PARAM – это выходные параметры действия , передающие в вызываемую форму значения, которые становятся значениями её входных параметров. Если в вызываемой форме описано значение по умолчанию входного параметра, то при совпадении кода (свойство NAME выходного параметра) это значение будет переопределено тем, которое указано в выходном параметре действия вызывающей формы.

Все выходные параметры, независимо от того – описаны они как входящие в вызываемой форме, будут доступны в ней по коду, указанному в свойстве NAME выходного параметра вызывающей формы.

В случае если набор выходных параметров действия не может быть заранее определён и становится известным только в момент выполнения действия, существует альтернативный способ передачи параметров: в виде коллекции, указанной в круглых скобках сразу после кода формы в свойстве COMMAND: `КОД_ФОРМЫ(Параметр1:Значение;Параметр2:Значение;...;)`. Возможен комбинированный вариант задания выходных параметров – и посредством подчинённых элементов действия типа PARAM и коллекцией параметров в свойстве действия COMMAND. Параметры, определённые отдельными элементами, имеют приоритет.

Code – ***Код выходного параметра.*** Уникальный код элемента среди прочих элементов формы. Задаётся при создании параметров. Код может быть изменён действием “Изменить” в навигаторе редактора форм.

Seq – ***Порядковый номер параметра.*** Задаёт порядок отображения параметров в навигаторе редактора форм. Может быть изменён действием “Изменить” в навигаторе редактора форм.

Name – ***Название параметра.*** Задаётся при создании. Используется для отображения в навигаторе редактора форм. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения. Может быть изменён действием “Изменить” в навигаторе редактора форм.

NAME – ***Код параметра***. Реальный код выходного параметра. Именно по нему заполняется значение входного параметра в вызываемой форме. Если код пустой, то данный выходной параметр не работает (ничего в вызываемую форму передаваться не будет).

Выходной параметр с кодом CONFIG открывает вызываемую форму в указанной тут конфигурации. При отсутствии его в выходных параметрах, вызываемая форма открывается в той же конфигурации, что и вызывающая. Если в вызываемой форме нет нужной конфигурации, она открывается в конфигурации DEFAULT.

VALUE – ***Значение выходного параметра***. При использовании динамики её значение вычисляется непосредственно в момент выполнения действия. При совпадении по коду с входным параметром вызываемой формы его значение по умолчанию переопределяется данным значением выходного параметра.

DESCRIPTION – ***Комментарий***. Пояснения разработчика о значении выходного параметра.

★ Оперативные свойства Списка и его элементов

- **{FormProperty:FormID}** – Уникальный в рамках сеанса текстовый идентификатор экземпляра формы. Не изменяется. Доступно в любом контексте.

- **{FormProperty:MetadataCode}** – Код метаданных, откуда загружался Список в память сервера приложений. Не изменяется. Доступно в любом контексте.
- **{FormProperty: MetadataApp}** – Идентифицирующий код приложения, из метаданных которого загружался список в память сервера приложений. Не изменяется. Доступно в свойствах Списка.
- **{FormProperty:Code}** – Идентифицирующий код формы, определяющий её сущность. В частности, к этому коду привязываются пользовательские настройки. При загрузке Списка из метаданных свойство {FormProperty:Code} получается из общего свойства Списка – FEATURE. Не изменяется. Доступно в свойствах Списка.
- **{FormProperty:ConfigCode}** – Код конфигурации метаданных, из которой загружался Список. Не изменяется. Доступно в любом контексте.
- **{FormProperty:ConfigName}** – Наименование конфигурации метаданных, из которой загружался Список (в текущем языке). «Виртуальная конфигурация», если Список чисто программный. Не изменяется. Доступно в любом контексте.
- **{FormProperty:FType}** – Тип формы = LIST. Не изменяется. Доступно в любом контексте.
- **{Property:Code}** – Код текущего элемента (в свойствах элемента). Не изменяется. Доступно в свойствах элементов.
- **{Property:EType}** – Тип текущего элемента (в свойствах элемента): ITEM, ACTION, PARAM или GROUP. Не изменяется. Доступно в свойствах элементов.
- **{Property:SelectionMode}** – Режим выделения строк в Списке: Single или Multi. Доступно в любом контексте.
- **{Property:Value}** – Текущее значение ячейки столбца. Доступно в свойствах столбцов и подчинённых им элементов (локальных действий и их параметров).
- **{Property:Parent}** – Код родительского элемента. Для самостоятельных элементов тут значение «Form». При некоторых обстоятельствах может изменяться! Доступно в свойствах элементов
- **{Property: CountSelected}** – Количество «особых» строк – Явно выделенных вне инверсии выделения или явно развыделенных в режиме инверсии выделения

★ Системные переменные уровня Списка

- **{Property:Var TransitParam}** – Коллекция значений параметров, указанных при загрузке метаданных в рабочую область. То есть это те параметры, с которыми открывалась форма, разработанная в метаданных
- **{Property:Var CurrentColumn}** – Код столбца, к которому относится текущая ячейка в текущей строке Списка
- **{Property:Var CurrentRow}** – Номер по порядку текущей строки Списка
- **{Property:Var FetchedRows}** – Количество извлечённых строк в гриде Списка на текущий момент.
- **{Property:Var Inversion}** – Режим инверсии выделения строк: Y – включён, все вновь извлекаемые строки изначально уже выделены. Любое другое значение (включая пустое) – выключен.
- **{Property:Var IsLastPage}** – Признак полноты полученных данных в Grid Списка: Y – все строки Списка уже извлечены. Любое другое значение – на сервере, возможно, есть ещё строки, не полученные в Grid.
- **{Property:Var Is_QBE_Mode}** – Режим Query by Example: Y – Grid Списка находится в режиме ввода условий QBE. Любое другое значение (включая пустое) – основной режим Grid.
- **{Property:Var MainQuery}** – Текст основного запроса Списка (с учётом условий QBE)
- **{Property:Var CurrentAction}** – Код текущего выполняющегося действия.
- **{Property:Var ParentFormID}** – ID рабочей области вызывающей формы.
- **{Property:Var UserConfigName}** – Наименование текущей пользовательской конфигурации
- **{Property:Var SysMenu}** – Необходимость генерации системного меню. Y-форма будет иметь системное меню. N-системное меню отсутствует.
- **{Property: Var ColumnMenu}** – Код столбца списка в котором последний раз было вызвано системное меню уровня столбца

Меню

LUI предоставляет возможность разработки и вызова иерархического меню. Как правило, прикладная система, созданная в LUI, имеет по крайней одно меню – главное меню, которое отображается пользователю сразу после входа в приложение. Форма меню строится по принципу иерархии (или дерева). Каждый узел дерева это действие, которое раскрывает подчинённые ему узлы и/или вызывает другие формы.

★ Общие свойства Меню

Code – ***Уникальный в рамках прикладной системы код формы***. Этот код вводится при создании меню и не меняется в разных его конфигурациях. По этому коду в совокупности с кодом конфигурации выполняется идентификация и загрузка формы меню из базы данных для выполнения в прикладной системе. Этот код указывается в действиях форм, которые вызывают меню. Уникальный код может быть изменён в редакторе форм LUI. При этом, автоматических изменений кода в действиях вызывающих форм не выполняется. Это надо делать вручную. Рекомендуется, чтобы код меню состоял из:

- Краткого обозначения прикладной системы
- Суффикса “M”
- Мнемоники отражающей смысл команды

Пример: TST_M_MAIN_MENU – главное меню в прикладной системе TEST

ON_LOAD – ***Код, выполняющийся при загрузке Меню***. Здесь могут размещаться команды процедурных языков, которые добавляют, удаляют или модифицируют свойства элементов Меню, загруженного из метаданных (см. раздел «Команды процедурных языков» на стр.15). Язык определяется префиксом. Программный код не нужно заключать в фигурные скобки, так как это не динамика, а, собственно, само значение свойства, что не исключает использование динамики для генерации всего кода или его части.

HEADER – ***Заголовок Меню***. Если бланк создаёт своё окно, то данный текст отображается как заголовок окна. Если это не первая форма в окне, то такой текст отображается как подзаголовок.

FORM_NOTES – ***Пояснения в Help***. Текст описания текущей конфигурации Бланка для отображения в Help и генерации документации.

★ Свойства входных параметров Меню

В меню, могут быть описаны входные параметры с единственной целью – указать значение параметров по умолчанию, так как при динамической ссылке на неопределённый входной параметр её значение останется неизвестным. Для ссылки на значение входного параметра укажите `{Param:КОД}` или `{F:КОД}`, где КОД – это код специального элемента меню типа PARAM - «Входной параметр». Следует заметить, что параметры всегда неявно обладают типом данных CHAR. Поэтому при кодировании свойств форм, запись ‘**{F:КОД}**’ всегда эквивалентна **{F2SQL:КОД}**.

Code – ***Код входного параметра.*** Задаётся при создании параметров. Именно по этому коду производится связывание выходных параметров действия в вызывающей форме с входными параметрами меню. Код параметра рекомендуется снабжать префиксом “P_”. Пример: P_CODE. ». Следует заметить, что в отличие от столбцов и полей списков и бланков параметры всегда неявно обладают типом данных CHAR. Поэтому при кодировании свойств форм, запись ‘**{F:КОД}**’ всегда эквивалентна **{F2SQL:КОД}**.

Name – ***Название входного параметра.*** Задаётся при создании параметров. Используется для отображения в навигаторе редактора форм. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

VALUE – ***Значение по умолчанию***. Если вызывающая Список форма не укажет значение параметра, то при ссылке на данный входной параметр в динамических свойствах будет подставляться указанное тут значение. По умолчанию – пустая строка. Значение по умолчанию может содержать динамические элементы. Вся динамика в значениях по умолчанию вычисляется в момент загрузки меню, а также в случае переинициализации формы после выполнения действия (см. свойство AFTER действий, значение REFRESH).

Если вызывающая форма переопределила значение по умолчанию входного параметра, то указываемая здесь динамика вычисляться никогда не будет.

DESCRIPTION – ***Описание***. Комментарий разработчика о входном параметре и о том, на что он влияет.

★ Неявные и системные входные параметры Меню

CONFIG – ***Код конфигурации меню.*** Если ни в выходных параметрах действия, которое вызывает меню, ни во входных параметрах самого меню не указан параметр CONFIG, то считается что он = DEFAULT. Т.е. будет загружаться конфигурация “по умолчанию” – DEFAULT.

WINDOW_TYPE$ – ***тип окна***. Этот параметр имеет значение для некоторых конфигураций сервера отображений. Он управляет свойствами визуализации и анимации главного меню. Возможные значения:

- Common –обычное окно (по умолчанию)
- MainMenu-окно с главным меню приложения

POSITION$ – ***положение окна.*** Этот параметр управляет положением окна с меню при его открытии. Возможные значения:

- Cascade – каскадом относительно родительского окна (по умолчанию);
- TopLeft – слева вверху экрана;
- WorkCenter-окно располагается в центре экрана;
- ParentRight-по-возможности вплотную справа от окна вызывающей формы;

WINWIDTH$ – ***ширина окна в пикселях.*** Этот параметр устанавливает ширину окна. Обычно сервер визуализации сам удачно подбирает ширину окна. Этот параметр позволяет разработчику отменить этот подбор. Этот параметр будет действовать, если ранее текущий пользователь ещё не изменял ширину окна данного меню, исходя из своих собственных предпочтений. Пример установки ширины окна равным 388 пикселей:

WINHEIGHT$ - ***высота окна в пикселях.*** Этот параметр устанавливает высоту окна. Обычно сервер визуализации сам удачно подбирает высоту окна. Этот параметр позволяет разработчику отменить этот подбор. Этот параметр будет действовать, если ранее текущий пользователь ещё не изменял высоту окна данного меню, исходя из своих собственных предпочтений.

CONECTION$ - ***код соединения с БД.*** Этот параметр следует применять в действии вызывающей формы для передачи кода соединения с БД и установки свойства Connection уровня формы. Если этот параметр не передан, то свойство Connection уровня формы будет унаследовано из свойства Connection уровня приложения. Значение свойства Connection может быть изменено в самом меню, например в свойстве ON_LOAD применена команда FormProperty:Connection=КОД_СОЕДИНЕНИЯ. Значение свойства Connection используется при выполнении команд **SQL:Команда**, вычислении динамических элементов **{SQL:Выражение}**.

★ Свойства пунктов Меню

Пунктами меню являются действия. Эти действия могут вызывать список подчинённых действий, вызывать другие формы, выполнять программный код, закрывать форму меню. Меню отображается в виде иерархии (дерева).

Code – ***Код действия.*** Уникальный код элемента среди прочих элементов формы. Задаётся при создании параметров. Код может быть изменён действием “Изменить” в навигаторе редактора форм.

Seq – ***Порядковый номер действия.*** Задаётся при создании. Это свойство задаёт положение пункта меню относительно других элементов расположенных на том же уровне. Порядковый номер можно изменять действием “Изменить” в навигаторе редактора форм. Или действиями “Вверх/Вниз” в бланке свойств элемента редактора форм.

Name – ***Название действия.*** Задаётся при создании. Используется для отображения в навигаторе редактора форм. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

Parent– ***Старший пункт.*** Задаётся при создании. Это свойство определяет вложенность элемента меню. Если пусто – элемент расположен на уровне формы. Может быть изменён действием “Изменить” в навигаторе редактора форм.

PROMPT – ***Наименование действия в интерфейсе пользователя***. Если приложение многоязычное, то рекомендуется задавать на всех языках приложения.

EXPLANATION – ***Подсказка***. Текст, поясняющий пункт меню одной ёмкой, но предельно точной фразой. Используется в виде всплывающей подсказки. Для детализации используйте свойство ACTION_NOTES. В ин-

терфейсе пользователя отображается, как правило, в виде текста возникающего рядом с узлом дерева. Если приложение многоязычное, то рекомендуется задавать на всех языках приложения.

ACTIVE – ***Признак активности***. Это свойство определяет можно ли выполнять данное действие. Это свойство не влияет на отображение и доступность подчинённых пунктов. Оно влияет на выполнение программного кода в свойствах PROC_INITIALLY, PROC_FINALLY и на вызов команды указанной в свойстве COMMAND. Возможные значения :

- Y (Да) - действие можно выполнить;
- N (Нет) - действие выполнить нельзя;
- Любое другое значение (включая пустое) эквивалентно значению N.

DISPLAY – ***Признак отображения.*** Это свойство определяет, следует ли отображать пункт меню в дереве.

Возможные значения:

- Y - Действие (пункт меню) отображается;
- N - Действие (пункт меню) не отображается;
- Любое другое значение (включая пустое) эквивалентно значению N

Следует иметь в виду, что если требуется отображение хотя бы одного подчинённого пункта, то отображаться будет и сам пункт, и все его старшие пункты вверх по иерархии.

ICON – ***Иконка***. Указывает имя иконки отображаемой в узле дерева меню.

AUTOEXPAND – ***Признак необходимости автоматического раскрытия***. Указывает, следует ли при отображении пункта меню раскрывать список подчинённых ему пунктов. Не действует вглубь иерархии. Действует только на непосредственно подчинённые пункты.

ROLE – ***Способы активации***. Альтернативные способы активации действия:

- DBLCLICK – Двойной клик левой кнопкой мыши или нажатие клавиши `Enter`;
- CLOSE – Нажатие клавиши `Esc` или закрытие окна. Выполняется также при попытке закрытия формы, каким бы способом это закрытие не выполнялось;
- AUTO – Действие, выполняющееся при каждом изменении значения любого его выходного параметра;
- RMB – Действие выполняемое путём выбора пункта меню возникающем по правой кнопке мыши.

По умолчанию считается, что ROLE= DBLCLICK, даже если значение не указано.

Если ROLE=CLOSE или ROLE=RMB пункт меню не будет отображаться в виде узла дерева. Действия с ролью CLOSE или ролью RMB должны располагаться только на уровне формы и не подчиняться другим действиям иначе они не будут выполнены. В случае наличия нескольких действий с ролью CLOSE выполняется активное действие с минимальным значением SEQ. В случае наличия нескольких активных действий с ролью RMB все они отображаются во всплывающем меню в порядке возрастания свойства Seq.

QUESTION – ***Текст предупреждения***. Если свойство не пусто, то его значение будет выдано как предупреждения с двумя кнопками. Если текст заканчивается знаком вопроса, то на кнопках будут надписи «Да» и «Нет». Иначе на кнопках отобразится «Продолжить» и «Отменить». Выбор первой кнопки позволяет продолжить выполнение действия. Выбор второй кнопки – приводит к прекращению выполнения действия и откату к началу его выполнения.

PROC_INITIALLY – ***Программный код действия***. Здесь указывается текст на процедурном языке, выполняющийся при активации действия перед (или вместо) вызовом формы, команды OS и т.п. Язык определяется префиксом (см. «Команды процедурных языков» на стр.15). Программный код не нужно заключать в фигурные скобки, так как это не динамика, а, собственно, само значение свойства, что не исключает использование динамики для генерации всего кода или его части.

COMMAND_TYPE – ***Тип вызываемого действия***. Возможны следующие значения:

- LIST – Вызов формы LUI. В этом случае свойство COMMAND – код списка или бланка из метаданных;
- BLANC – Вызов формы LUI. В этом случае свойство COMMAND – код списка или бланка из метаданных;

- TREE - Вызов иерархической структуры LUI. В этом случае свойство COMMAND – код списка из метаданных, который образует элементы первого уровня иерархической структуры;
- HOST – Вызов внешней команды OS на сервере приложений. В этом случае свойство COMMAND – команда OS из списка явно разрешённых команд;
- URL – Вызов WEB-страницы на клиентском устройстве. В этом случае свойство COMMAND – URL страницы;
- MENU –Вызов формы LUI. В этом случае свойство COMMAND – код формы LUI типа «Меню»;
- TASK – Вызов бланка запуска Задачи LUI. В этом случае свойство COMMAND – код Задачи, запускающейся в синхронном режиме.

COMMAND – ***Объект действия***. Вызываемый объект, соответствующий типу действия, указанному в свойстве COMMAND_TYPE. Помимо традиционного способа передачи параметров (отдельными подчинёнными элементами действия типа PARAM), возможно перечисление параметров в коллекции, указанной в круглых скобках сразу после объекта - идентификатора действия:

```
Объект(Параметр1:Значение;Параметр2:Значение;...;).
```

PROC_FINALLY – ***Программный код после действия***. Здесь указывается текст на процедурном языке, выполняющийся после вызова и завершения формы, задачи, команды OS и т.п. Язык определяется префиксом. Программный код не нужно заключать в фигурные скобки, так как это не динамика, а, собственно, само значение свойства, что не исключает использование динамики для генерации всего кода или его части.

Можно последовательно указать несколько программ на разных процедурных языках. В этом случае языки должны отделяться друг от друга как минимум одной пустой строкой.

В случае если свойство APPLY_TO = PROGRESS, данный программный код (коды) выполняется для каждой выделенной строки.

Если вызываемая командой COMMAND форма (Список, Бланк или Иерархическая структура) закрылась в режиме CANCEL (действие в вызываемой форме, выполняющееся по закрытию, имеет свойство AFTER = CANCEL), то код из PROC_FINALLY не выполняется, действие отменяется, а транзакция откатывается к началу действия.

CHILD_COMMIT –***Разрешать завершать транзакцию в вызываемой форме?*** В настоящий момент не действует.

AFTER – ***Поведение после выполнения действия***. После успешного выполнения действия в случае, если вызывающая форма не закрылась в режиме CANCEL, меню поведёт себя в соответствии с указанными тут вариантами:

- NONE – Ничего не произойдёт;
- REFRESH – Переинициировать Меню, то есть, как бы выйти и войти с теми же значениями входных параметров. Код из свойства ON_LOAD выполнится повторно;
- EXIT – Закрыть и передать управление в вызывающую форму. В случае визарда закроется весь визард;
- CANCEL – Отменить вызов Меню. В вызывающей форме не будут выполняться POST-действия и перезапросы. В случае визарда откатится весь визард. Для отката только на предыдущий шаг визарда данное действие должно иметь ROLE = WIZARD.

Любое другое значение, включая пустую строку, эквивалентно NONE.

ACTION_NOTES – ***Пояснения в Help***. Текст описания действия (пункта меню) для отображения в Help и генерации документации. Следует помещать сюда всё, что не уместилось в одну фразу в свойстве EXPLANATION. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

★ Свойства выходных параметров действия пункта Меню

Подчинённые действию элементы типа PARAM – это выходные параметры действия, передающие в вызываемую форму значения, которые становятся значениями её входных параметров. Если в вызываемой форме описано значение по умолчанию входного параметра, то при совпадении кода (свойство NAME выходного параметра) это значение будет переопределено тем, которое указано в выходном параметре действия вызывающей формы.

Все выходные параметры, независимо от того – описаны они как входящие в вызываемой форме, будут доступны в ней по коду, указанному в свойстве NAME выходного параметра вызывающей формы.

В случае если набор выходных параметров действия не может быть заранее определён и становится известным только в момент выполнения действия, существует альтернативный способ передачи параметров: в виде коллекции, указанной в круглых скобках сразу после кода формы в свойстве COMMAND: `КОД_ФОРМЫ(Параметр1:Значение;Параметр2:Значение;...;)`. Возможен комбинированный вариант задания выходных параметров – и посредством подчинённых элементов действия типа PARAM и коллекцией параметров в свойстве действия COMMAND. Параметры, определённые отдельными элементами, имеют приоритет.

Code – ***Код выходного параметра.*** Уникальный код элемента среди прочих элементов формы. Задаётся при создании параметров. Код может быть изменён действием “Изменить” в навигаторе редактора форм.

Seq – ***Порядковый номер параметра.*** Задаёт порядок отображения параметров в навигаторе редактора форм. Может быть изменён действием “Изменить” в навигаторе редактора форм.

Name – ***Название параметра.*** Задаётся при создании. Используется для отображения в навигаторе редактора форм. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения. Может быть изменён действием “Изменить” в навигаторе редактора форм.

NAME – ***Код параметра***. Реальный код выходного параметра. Именно по нему заполняется значение входного параметра в вызываемой форме. Если код пустой, то данный выходной параметр не работает (ничего в вызываемую форму передаваться не будет).

Выходной параметр с кодом CONFIG открывает вызываемую форму в указанной тут конфигурации. При отсутствии его в выходных параметрах, вызываемая форма открывается в той же конфигурации, что и вызывающая. Если в вызываемой форме нет нужной конфигурации, она открывается в конфигурации DEFAULT.

VALUE – ***Значение выходного параметра***. При использовании динамики её значение вычисляется непосредственно в момент выполнения действия. При совпадении по коду с входным параметром вызываемой формы его значение по умолчанию переопределяется данным значением выходного параметра.

DESCRIPTION – ***Комментарий***. Пояснения разработчика о значении выходного параметра.

★ Оперативные свойства Меню и его элементов

Большинство свойств совпадает со свойствами списка. Особенностями являются отсутствие свойств, присущих только списку, таких как SelectionMode, CountSelected и др. Кроме того, имеются свойство специфичное для меню, {Property:CurrentMenuItem} – текущий пункт меню.

Бланк

Форма-бланк – это набор элементов-полей ввода данных, над которыми могут производиться некоторые действия. Каждый элемент бланка имеет свой набор свойств.

Поля в бланках могут объединяться в группы, которые в свою очередь могут включаться в другие группы. Теоретически глубина иерархии группировки элементов формы-бланка не ограничена. Порядок следования полей и групп одного уровня вложенности определяется только их номером по порядку. Группы имеют свои свойства, основное из которых – тип. Тип группы определяет способ её отображения и расположение элементов внутри неё: Блок закладок, сворачивающаяся и разворачивающаяся область, элементы, объединённые общим заголовком и т.п.

Каждое поле бланка имеет статическое или вычисляемое значение по умолчанию. Помимо этого, для поля можно указать процедуру автоматического обновления значения. Автоматическое обновление значения поля происходит при каждом изменении любого поля, от значения которого оно зависит (прямо или косвенно).

Контекстом для вычисления любых динамических свойств любых элементов бланка являются текущие значения всех полей. Перевычисление некоторого динамического свойства происходит в момент изменения значения поля. Например, после заполнения некоторого поля может открыться новая закладка или появится дополнительные поля или измениться список допустимых значений другого поля, стать активной кнопка и т.п.

Значения полей (если это позволено) могут меняться пользователем разными способами: свободный ввод с клавиатуры, выбор из списка допустимых или типичных значений, переключателями и т.п. Проверка корректности данных в процессе редактирования полей может производиться, но не должно вызывать ошибок и исключений. Окончательная проверка может быть произведена только перед или в процессе выполнения некоторого действия в бланке, например, нажатия кнопки.

Поля бланков могут поддерживать полнофункциональный редактор с форматированием текста и шрифта, вставкой мультимедиа объектов.

Действия в формах-бланках практически эквивалентны действиям в формах-списках, только с иным контекстом (там – значения полей меняющейся текущей строки, тут – изменяемые значения всех полей). Действия, как и в списках, могут активироваться различными способами: выбором пункта из локального меню, нажатием кнопки с текстом, нажатием кнопки с пиктограммой в поле... Кроме того, действия могут быть привязаны и выполняться автоматически при событиях: нажатие ролевой клавиши (такой, как Enter…), соответствие текста в поле регулярному выражению, закрытии формы, по таймеру и т.п.

Динамические свойства при вычислении элементов формы-бланка могут опираться на публичные индикаторы и флаги формы, например: Код текущего поля, код выполняемого действия, признак наличия изменений в поле или во всём бланке и т.п.

★ Общие свойства Бланка

Code – *Уникальный в рамках прикладной системы код формы*. Этот код вводится при создании бланка и не меняется в разных его конфигурациях. По этому коду в совокупности с кодом конфигурации выполняется идентификация и загрузка формы бланка из базы данных для выполнения в прикладной системе. Этот код указывается в действиях форм, которые вызывают бланк. Уникальный код может быть изменён в редакторе форм LUI. При этом, автоматических изменений кода в действиях вызывающих форм не выполняется. Это надо делать вручную. Рекомендуется, чтобы код списка состоял из:

- Краткого обозначения прикладной системы
- Суффикса “B”
- Мнемоники отражающей смысл команды

Пример: TST_B_DEPT – бланк ввода и редактирования данных об отделе в прикладной системе TEST

QUERY_TEXT - ***Select заполнения***. SQL-запрос, выполняющийся при построении бланка на основе метаданных, который опирается только на значения входных параметров бланка. Если запрос возвращает одну строку, то значения столбцов первой строки станут либо значениями по умолчанию перечисленных полей бланка (если псевдоним (алиас) столбца совпадает с кодом соответствующего поля), либо значениями виртуальных параметров бланка, на которые могут ссылаться все свойства всех элементов бланка (если алиас столбца не совпадает ни с одним кодом элемента бланка).

Если запрос не возвращает ни одной строки, то будут действовать значения по умолчанию полей, указанные в их свойствах DEFINITION.

Если запрос возвращает более одной строки, то поля, для которых все значения в возвращаемых строках совпадают, заполнятся этим значением, а поля, для которых значения различаются – становятся пустыми (`{Property:Value}` – пусто), но зато свойство `{Property:Diff}` становится равным Y. На основании этого можно выделять стилями поля, значения которых не определено.

В самом начале, перед Select, в круглых скобках может быть указан код соединения (если в Приложении используется более одного соединения с базами данных). Если код соединения не будет указан, то будет использоваться соединение, код которого к моменту выполнения запроса хранится в свойстве формы **Connection**. Это свойство формы наследуется из приложения или передаётся в параметре CONNECTION$ и может быть изменено самой формой. См. описание команды DBConnect – подключение к базе данных.

ON_LOAD – *Код, выполняющийся при загрузке Бланка*. Здесь могут размещаться команды процедурных языков, которые добавляют, удаляют или модифицируют свойства элементов Бланка, загруженного из метаданных (см. раздел «Команды процедурных языков» на стр.15). Язык определяется префиксом. Программный код не нужно заключать в фигурные скобки, так как это не динамика, а, собственно, само значение свойства, что не исключает использование динамики для генерации всего кода или его части.

HEADER – *Заголовок Бланка*. Если бланк создаёт своё окно, то данный текст отображается как заголовок окна. Если это не первая форма в окне, то такой текст отображается как подзаголовок

FORM_NOTES – ***Пояснения в Help***. Текст описания текущей конфигурации Бланка для отображения в Help и генерации документации.

FEATURE – ***Код привязки***. Это рабочий код Бланка. Используется для ассоциации с ним пользовательских интерфейсных предпочтений, а также прав доступа к элементам для групп пользователей. По умолчанию код привязки совпадает с кодом метаданных, из которых был загружен Бланк

★ Свойства входных параметров Бланка

В бланке, могут быть описаны входные параметры с единственной целью – указать значение параметров по умолчанию, так как при динамической ссылке на неопределённый входной параметр её значение останется неизвестным. Для ссылки на значение входного параметра укажите `{Param:КОД}` или `{F:КОД}`, где КОД – это код специального элемента меню типа PARAM - «Входной параметр». Следует заметить, что параметры всегда неявно обладают типом данных CHAR. Поэтому, при кодировании свойств форм запись '**{F:КОД}**' всегда эквивалентна **{F2SQL:КОД}**.

Code – ***Код входного параметра.*** Задаётся при создании параметров. Именно по этому коду производится связывание выходных параметров действия в вызывающей форме с входными параметрами меню. Код параметра рекомендуется снабжать префиксом "P_". Пример: P_CODE.

Name – ***Название входного параметра.*** Задаётся при создании параметров. Используется для отображения в навигаторе редактора форм. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

VALUE – ***Значение по умолчанию***. Если вызывающая Бланк форма не укажет значение параметра, то при ссылке на данный входной параметр в динамических свойствах будет подставляться указанное тут значение. По умолчанию – пустая строка. Значение по умолчанию может содержать динамические элементы. Вся динамика в значениях по умолчанию вычисляется в момент загрузки бланка, а также в случае переинициализации формы-бланка после выполнения действия (см. свойство AFTER действий, значение REFRESH)

Если вызывающая форма переопределила значение по умолчанию входного параметра, то указываемая здесь динамика вычисляться никогда не будет.

DESCRIPTION – ***Описание***. Комментарий разработчика о входном параметре и о том, на что он влияет.

★ Неявные и системные входные параметры Бланка

CONFIG – ***Код конфигурации бланка.*** Если ни в выходных параметрах действия, которое вызывает бланк, ни во входных параметрах самого бланка не указан параметр CONFIG, то считается что он = DEFAULT. Т.е. будет загружаться конфигурация "по умолчанию" – DEFAULT.

WINDOW_TYPE$ – ***тип окна***. Этот параметр имеет значение для некоторых конфигураций сервера отображений. Он управляет свойствами визуализации окна. Возможные значения:

- Common – обычное окно (по умолчанию)
- Logon - окно входа в приложения

POSITION$ – ***положение окна.*** Этот параметр управляет положением окна бланка при его открытии. Возможные значения:

- Cascade – каскадом относительно родительского окна (по умолчанию)
- TopLeft – слева вверху экрана
- WorkCenter-окно располагается в центре экрана

WINWIDTH$ – ***ширина окна в пикселях.*** Этот параметр устанавливает ширину окна. Обычно сервер визуализации сам удачно подбирает ширину окна. Этот параметр позволяет разработчику отменить этот подбор. Этот параметр будет действовать, если ранее текущий пользователь ещё не изменял ширину окна данного бланка, исходя из своих собственных предпочтений.

WINHEIGHT$ - ***высота окна в пикселях.*** Этот параметр устанавливает высоту окна. Обычно сервер визуализации сам удачно подбирает высоту окна. Этот параметр позволяет разработчику отменить этот подбор. Этот параметр будет действовать, если ранее текущий пользователь ещё не изменял высоту окна данного бланка, исходя из своих собственных предпочтений.

CONECTION$ - ***код соединения с БД.*** Этот параметр следует применять в действии вызывающей формы для передачи кода соединения с БД и установки свойства Connection уровня формы. Если этот параметр не передан, то свойство Connection уровня формы будет унаследовано из свойства Connection уровня прило-

жения. Значение свойства Connection может быть изменено в самом бланке, например в свойстве ON_LOAD применена команда FormProperty:Connection=КОД_СОЕДИНЕНИЯ, или в свойстве QUERY_TEXT перед тестом запроса указа код соединения в круглых скобках. Значение свойства Connection используется при выполнении команд **SQL:Команда**, вычислении динамических элементов **{SQL:Выражение}**, при выполнении основного запроса списка, если не указан код соединения в круглых скобках.

★ Свойства полей Бланка

Поля – это области формы, где отображается и/или может быть введена или изменена информация. У каждого поля может быть надпись, кратко поясняющая суть информации, а также набор локальных действий, доступных только в этом поле. Информация может меняться различными способами: вводом с клавиатуры, выбором значения из списка предложенных вариантов, последовательным перебором допустимых значений, посредством выполнения некоторого действия. Данные в полях могут ещё обновляться автоматически (так же как и любые свойства любых элементов), если указать алгоритм пересчёта, зависящий от других полей того же Бланка. В конечном итоге, значения полей используются в некотором действии – для передачи их в другие формы интерфейса, для операций в БД, для операций с файлами доступными серверу приложений, для выполнения команд ОС сервера приложений и т.д. Динамические свойства элементов бланка могут ссылаться на значение полей. Для этого укажите **{F:КОД}** или **{F2SQL:КОД}**, где КОД – это код элемента типа «Поле».

Code – ***Код поля.*** Уникальный идентификатор элемента в форме. Вводится при создании поля редактором форм. Значение остаётся неизменным во всех конфигурациях списка. Может быть изменено действием “Изменить” в навигаторе редактора форм.

ITEM_TEMPLATE – ***Шаблон элемента***. Это свойство задаётся при создании поля и имеет постоянное значение во всех конфигурациях бланка. Обычно оно автоматически заполняется редактором форм, если для указанного типа данных в шаблонном бланке BLANK_TEMPLATE имеется столбец с кодом **FIELD_тип данных**. Шаблон определяет свойства поля “по умолчанию”. Значение может быть изменено действием “Изменить” в навигаторе редактора форм.

Seq – ***Порядковый номер элемента.*** Это свойство задаётся при создании поля и имеет постоянное значение во всех конфигурациях бланка. Может быть изменено действием “Изменить” в навигаторе редактора форм. Порядковый номер определяет положение поля среди остальных полей той же группы полей.

Name – ***Название элемента.*** Это свойство задаётся при создании поля и имеет постоянное значение во всех конфигурациях бланка. Может быть изменено действием “Изменить” в навигаторе редактора форм. Название используется для отображения в навигаторе редактора форм и для документации.

Parent – ***Родительская группа.*** Это свойство задаётся при создании поля и имеет постоянное значение во всех конфигурациях списка. Может быть изменено действием “Изменить” в навигаторе редактора форм. Поле в бланке располагается либо вне групп, либо в одной из групп полей.

DATA_TYPE – ***Тип данных поля***. Это свойство поля задаётся при его создании и имеет постоянное значение во всех конфигурациях бланка.

Указанный тип используется в процедуре конвертации полученных значений поля по умолчанию при выполнении SQL-запроса из свойства QUERY_TEXT. Кроме того, указанный тип используется для автокоррекции введённого пользователем значения и приведения его к указанному типу по указанному формату. Помимо этого, при выполнении действия может производиться проверка на соответствие значения поля указанному типу и указанному формату (свойство FORMAT). Тип данных используется в операторе языка LUI F2SQL для формирования конструкции пригодной для применения в языке SQL.

Архитектор прикладной системы может добавлять новые типы данных и изменять свойства существующих типов данных. Поставляемые в составе LUI типы данных:

- BOOLEAN – логический тип данных. Возможные значения: Y-истина;N-ложь, NULL – пусто;
- CHAR – Текстовая строка (значение по умолчанию).
- CIDR – IP-сеть (на данный момент поддерживается только стандарт v4)
- DATETIME – Дата и/или время. Настоятельно рекомендуется использование формата данных (свойство FORMAT);
- DTRANGE – диапазон даты/времени. Составной тип, имеющий четыре части
    - Признак вхождения в диапазон нижней границы
    - Нижняя граница – дата/время
    - Верхняя граница – дата/время

    - Признак вхождения в диапазон верхней границы

Существует специальное значение “empty” - пустой диапазон (не путать с NULL!) отображаемое как “()”.

Нижняя и верхняя границы могут отсутствовать. Это означает бесконечность снизу или сверху.

Настоятельно рекомендуется использование формата данных (свойство FORMAT). Формат применяется при извлечении данных из БД к каждой границе.

- INET - IP-адрес (на данный момент поддерживается только стандарт v4)
- INTERVAL – продолжительность во времени. Настоятельно рекомендуется использование формата данных (свойство FORMAT);
- MACADDR – MAC-адрес 6-ти байтная версия;
- MACADDR8 – MAC-адрес 8-ми байтная версия;
- NUMBER – Число. Возможно использование формата данных (свойство FORMAT);
- NUMRANGE – диапазон чисел. Составной тип, имеющий четыре части
    - Признак вхождения в диапазон нижней границы
    - Нижняя граница – число
    - Верхняя граница – число
    - Признак вхождения в диапазон верхней границы

Существует специальное значение “empty” - пустой диапазон (не путать с NULL!) отображаемое как “()”.

Нижняя и верхняя границы могут отсутствовать. Это означает бесконечность снизу или сверху.

Возможно использование формата данных (свойство FORMAT). Формат применяется при извлечении данных из БД к каждой границе.

- NLSTEXT – Многоязычный текст. Каждый используемый язык может отображаться в отдельном поле;
- RICHTEXT – Форматированный текст. В поле можно будет вставлять мультимедиа элементы, а также форматировать текст посредством текстового редактора;

DEFINITION – ***Значение по умолчанию***. Это значение (статическое или вычисляемое динамически) помещается в поле в момент открытия бланка только в том случае, если SQL-запрос из свойства QUERY_TEXT бланка не возвращает строк или в нём нет столбца с алиасом, совпадающим с кодом поля. В противном случае значение этого свойства будет проигнорировано, а значением по умолчанию станет значение столбца с алиасом, совпадающим с кодом этого поля Бланка.

RENOVATION – ***Перевычисление поля***. Здесь в фигурных скобках указывается алгоритм автоматического изменения значения поля на декларативном языке. Язык определяется префиксом. Указанная динамика срабатывает и автоматически заменят значение поля только в том случае, если изменится какое-то поле, явно используемое в динамическом выражении.

FORMAT – ***Формат поля***. Для типов данных поддерживающих формат здесь указывается формат преобразования к строке. Для типов данных поддерживающих алфавит здесь указывается алфавит проверки значения поля. Например:

- Если DATA_TYPE = CHAR, тут можно указать код алфавита, на соответствие которому будет проверяться или автокорректироваться вводимый текст;
- В случае DATA_TYPE = NUMBER или DATETIME, формат указывается согласно документации по СУБД;

Указанный формат или алфавит может использоваться при проверке и автокоррекции введённых данных, а также при заполнении значения посредством SQL-запроса из свойства QUERY_TEXT Бланка.

VERIFIER – ***Верификатор.*** Указывается ссылка на форму верификатора. Верификатор позволяет контролировать правильность ввода данных в поле и изменять содержимое поля “на лету” используя JavaScript браузера пользователя. Кроме того, верификатор может иметь программу проверки введённого значения, выполняемую на сервере приложений LUI или сервере БД.

CAPTION – ***Подпись к полю***. Текстовая надпись слева или сверху от поля, отражающая суть информации в данном поле.

VISIBLE – ***Количество отображаемых строк***. Возможные значения:

- 1 (или Y) - Однострочное поле;
- 2 и более - многострочное поле. Возможно добавление перевода строк;

- Любое другое значение (включая пустую строку) означает невидимое (неотображаемое) поле.

VISUALIZATION – ***Код набора свойств отображения***. Способ визуально выделить поле. По сути это – имя класса для стилевого оформления поля. Автоматически поддерживаются следующие варианты:

- FONT_SELECTED – Выделен текст;
- BACKGROUND_SELECTED – Выделен фон;
- ALL_SELECTED – Выделен текст и фон;
- LOWERED – слабозаметный;
- DANGER – Опасность!;
- ATTENTION – Внимание!;
- OK – Успех;
- INHERIT – Унаследованное значение;
- SECTION – Подраздел;
- START – Включить/Работает;
- INTERIM – Приостановить/Переключается;
- STOP – Остановить/Выключено.
- NOTES – Пояснения, дополнительные сведения

MAKEUP – ***Выражение для подсветки фрагментов текста в ячейках***. Здесь можно ввести одно или более регулярных выражений для применения стилей визуализации к подстрокам соответствующим этим выражениям. Пример:
если в это поле введено /(d{4})\.(d{4})\.(d{2})\.(d{2}) и в свойстве "Класс отображения" указано NORMAL DANGER INHERIT OK, и поле содержит 2019.07.01, то всё поле будет отображаться классом NORMAL, год - классом DANGER, месяц - классом INHERIT, день - классом OK

HINT – ***Подсказка***. Текст, поясняющий суть информации в поле одной ёмкой, но предельно точной фразой.

VARIANTS – ***Набор возможных значений***. Свойство служит для получения перечня возможных или типичных значений данного поля. Здесь можно указать:

- SQL - запрос. Будут отображаться только столбцы с алиасами в двойных кавычках. Значением поля станет значение из первого столбца выбранной строки;
- Коллекцию вариантов в формате Значение1:Пояснение;Значение2:Пояснение;... Выбранный результат – значение выбранного варианта

Запрос и коллекцию не нужно заключать в фигурные скобки, ибо это не динамика, а, собственно, значение свойства, что не исключает использование динамики для генерации запроса или коллекции

Первый вариант отображается в виде модального элемента выбора. Второй вариант отображается в виде выпадающего списка у поля бланка.

Система сама может разобраться, какой вариант подразумевал разработчик. Но в сложных случаях рекомендуется делать подсказки в виде префиксов LOVQ: и LOVD:.

LIVESEARCH– ***Запрос для "живого" поиска***. Здесь можно ввести одностолбцовый запрос к БД , который будет выполняться по мере ввода данных пользователем. Результат запроса будет отображаться под или над полем в виде списка значений доступных для выбора, создавая эффект динамической подсказки или помощника ввода данных. Выбираемое значение будет введено в поле.

Пример: (MY_CONNECT)select value from my_table where value ilike '{Property:Value}%'. Если в таблице my_table содержатся 'Иванов','Петров','Сидоров','Самсонов','Иваненко' и пользователь введёт в пустое поле букву "и", то будут отображены для выбора первая и последняя фамилии этого списка.

PATTERN – ***Условие автоперехода***. Здесь ожидается шаблон регулярного выражения. Ввод пользователя постоянно проверяется на соответствие этому шаблону. Автопереход на следующее поле случается при первом же соответствии введённых данных заданному здесь шаблону. Автопереход можно подменить выполнением некоего локального действия этого поля, у которого свойство ROLE = MATCH.

CHANGING – ***Способ изменения данных***. Способ ввода информации в поле. Возможны следующие значения:

- Y – Произвольный текст (значение по умолчанию). Пользователь может вводить данные с клавиатуры;

- C – Список значений. Пользователь может делать выбор только из списка доступных значений из свойства VARIANTS;
- U – Текст в верхнем регистре. Вводимый с клавиатуры текст сразу вводится в верхнем регистре;
- H - Скрытый текст для ввода паролей. Вводимые символы визуально не различимы;
- B – Check Box. Пользователь переключает значения из списка, определённого свойством VARIANTS;
- Любое другое значение (включая пустую строку) интерпретируется как неизменяемое поле;

Если при значениях Y и U определено свойство VARIANTS, то возможен выбор из наиболее типичных значений.

SHOW – ***Расшифровка значения***. При CHANGING равном C и N данное свойство подменяет реальное значение поля всегда. Во всех других случаях - только когда курсор не в этом поле. Если свойство не указано (пусто), то работает стандартный механизм вычисления расшифровки Read-Only и LOV-Only полей, а именно - расшифровка вычисляется исходя из множества значений, указанном в свойстве VARIANTS

OPTIONAL – ***Можно не заполнять?*** Возможны следующие значения:

- N – в поле должно отображаться непустое значение;
- Любое другое значение свойства (включая пустую строку) - поле можно не заполнять.

По умолчанию значение Y. Надо понимать, что на не пустоту проверяется значение свойства SHOW, так как именно это значение видит пользователь в поле.

ITEM_NOTES – ***Пояснения в Help***. Текст описания поля Бланка для отображения в Help и генерации документации. Следует помещать сюда всё, что не уместилось в одну фразу в свойстве HINT.

★ Свойства действий Бланка

Любое действие бланка выполняется в контексте текущих значений полей бланка. Корректность данных в полях жёстко проверяется только непосредственно перед- или в процессе выполнения действия. По результатам проверки может быть возбуждена ошибка с откатом состояния к началу выполнения действия.

Действия могут активироваться нажатием кнопок, выбором пункта из локального меню поля, кликом на иконке справа от поля. Также некоторые действия могут выполняться автоматически или по событию (Enter, Esc, закрытие формы и т.п.)

PROMPT – ***Наименование действия***. Этот текст отображается или на кнопке (для глобального действия) или в локальном меню (для локального действия поля), если, конечно, PROMPT не пустой.

В тексте можно использовать вертикальную черту. Тогда текст до вертикальной черты будет образовывать узел в локальном меню, а текст после черты – текст пункта подменю, раскрываемом подузлом. Глубина вложенности иерархии меню не ограничена.

В любое место текста можно вставить конструкцию `[ico:`***`Имя`***`]` где «Имя» - название иконки. В этом случае прямо в данном месте текста при выводе локального меню будет отображаться указанная иконка.

EXPLANATION – ***Подсказка***. Текст, поясняющий суть действия одной ёмкой, но предельно точной фразой. Используется в виде всплывающей подсказки у управляющего элемента, активирующего действие.

ACTIVE – ***Признак активности***. Это свойство проверяется как минимум 2 раза. Первый – при отображении элемента управления, активирующего данное действие (активно/не активно), второй – непосредственно перед выполнением. Возможные значения:

- Y – Действие доступно для выполнения любым возможным способом активации (по умолчанию);
- Любое другое значение (включая пустую строку) – Действие не доступно для выполнения, каким бы способом оно ни активировалось;

DISPLAY – ***Отображать кнопку?*** Создавать ли кнопочку для активации глобального действия. Если действие не активно, но создаёт кнопку, то эта кнопка будет не активна. Возможные значения:

- Y - Действие порождает кнопку (при наличии непустого PROMPT);
- Любое другое значение (включая пустую строку) – Действие не порождает кнопку.

Значение по умолчанию синхронизировано со свойством ACTIVE (ссылается на значение этого свойства: `{Property:ACTIVE}`).

ICON – ***Иконка***. Имя иконки, активирующей локальное действие (подчинённое полю). Иконка всегда будет видимой. Для сокрытия иконки сделайте значение данного свойства динамическим. Активность иконки изменяется в соответствии со свойством ACTIVE. Если несколько локальных действий имеют одну иконку, то она будет отображаться только один раз. При нажатии на эту иконку из них сработает первое активное действие.

У поля могут появляться системные локальные действия со своими иконками:

- Calendar – Если поле типа «Дата/Время» (DATA_TYPE = DATETIME)
- LOV – Если свойство поля VARIANTS содержит SQL-запрос для отображения List of Value
- DropDown – Если свойство поля VARIANTS содержит коллекцию вариантов для выпадающего списка
- edit – Если свойство поля DATA_TYPE = RICHTEXT

Согласно правилам вызова действий, при нажатии иконки любое локальное действие, если его иконка совпадает с системной, может переопределить вызов соответствующего стандартного действия при нажатии на эту иконку.

ROLE – ***Способы активации***. Альтернативные способы активации действия:

- DBLCLICK – выполняется при нажатии `Enter` в любом однострочном поле бланка;
- CLOSE – Нажатие клавиши `Esc` или закрытие окна. Выполняется также при попытке закрытия формы, каким бы способом это закрытие не выполнялось;
- AUTO – Действие, выполняющееся при каждом изменении значения любого его выходного параметра;
- TIME – Действие, выполняющееся ежесекундно по таймеру (в периоды активности формы);
- WIZARD – Действие, являющееся очередным шагом визарда (вперёд или назад);
- MATCH – Для локальных действий в полях. Активируется при соответствии значения шаблону из свойства PATTERN этого родительского поля.

Каждый способ активации выполняет первое по порядку активное действие, которому назначен данный способ активации. Способы активации можно комбинировать, указывая их через пробел и/или запятую.

QUESTION – ***Текст предупреждения***. Если свойство не пусто, то его значение будет выдано как предупреждения с двумя кнопками. Если текст заканчивается знаком вопроса, то на кнопках будут надписи «Да» и «Нет». Иначе на кнопках отобразится «Продолжить» и «Отменить». Выбор первой кнопки позволяет продолжить выполнение действия. Выбор второй кнопки – приводит к прекращению выполнения действия и откату к началу его выполнения.

VERIFICATION – ***Контролировать правильность значений?*** Некоторую проверку корректности ввода данных в редактируемых видимых полях Бланка может взять на себя интерфейс, а именно: заполненность обязательных полей и соответствие значений полей типу и формату данных:

- Y (по умолчанию) – проверять поля на заполненность и соответствие типу и формату;
- Любое другое значение (включая пустое) – не производить проверку. Например, при закрытии формы без сохранения данных.

Более сложную проверку корректности данных в поля должен брать на себя код из свойства PROC_INITIALLY.

PROC_INITIALLY – ***Программный код действия.*** Здесь указывается программный код на процедурном языке, выполняющийся при активации действия перед (или вместо) вызовом формы, задачи, команды OS и т.п. Язык определяется префиксом (см. раздел «Команды процедурных языков» на стр.15). Программный код не нужно заключать в фигурные скобки, так как это не динамика, а, собственно, само значение свойства, что не исключает использование динамики для генерации всего кода или его части.

Можно последовательно указать несколько программ на разных процедурных языках. В этом случае языки должны отделяться друг от друга как минимум одной пустой строкой. Следствие: внутри кода процедурного языка не должно быть пустых строк.

COMMAND_TYPE – ***Тип вызываемого действия***. Возможны следующие значения:

- LIST – Вызов формы LUI. В этом случае свойство COMMAND – код списка или бланка из метаданных;

- BLANC – Вызов формы LUI. В этом случае свойство COMMAND – код списка или бланка из метаданных;
- TREE - Вызов иерархической структуры LUI. В этом случае свойство COMMAND – код списка из метаданных, который образует элементы первого уровня иерархической структуры;
- HOST – Вызов внешней команды OS на сервере приложений. В этом случае свойство COMMAND – команда OS из списка явно разрешённых команд;
- URL – Вызов WEB-страницы на клиентском устройстве. В этом случае свойство COMMAND – URL страницы;
- MENU –Вызов формы LUI. В этом случае свойство COMMAND – код формы LUI типа «Меню»;
- TASK – Вызов бланка запуска Задачи LUI. В этом случае свойство COMMAND – код Задачи, запускающейся в синхронном режиме.

COMMAND – ***Объект действия***. Вызываемый объект, соответствующий типу действия, указанному в свойстве COMMAND_TYPE. Помимо традиционного способа передачи параметров (отдельными подчинёнными элементами действия типа PARAM), возможно перечисление параметров в коллекции, указанной в круглых скобках сразу после объекта - идентификатора действия: `Объект(Параметр1:Значение;Параметр2:Значение;...;)`. Даже при обработке действием нескольких строк вызов указанного тут объекта выполняется только 1 раз.

PROC_FINALLY – ***Программный код после действия***. Здесь указывается текст на процедурном языке, выполняющийся после вызова и завершения формы, задачи, команды OS и т.п. Язык определяется префиксом. Программный код не нужно заключать в фигурные скобки, так как это не динамика, а, собственно, само значение свойства, что не исключает использование динамики для генерации всего кода или его части.

Можно последовательно указать несколько программ на разных процедурных языках. В этом случае языки должны отделяться друг от друга как минимум одной пустой строкой.

Если вызываемая командой COMMAND форма (Список, Бланк или Иерархическая структура) закрылась в режиме CANCEL (действие в вызываемой форме, выполняющееся по закрытию, имеет свойство AFTER = CANCEL), то код из PROC_FINALLY не выполняется, действие отменяется, а транзакция откатывается к началу действия.

CHILD_COMMIT – ***Разрешать завершать транзакцию в вызываемой форме?*** В настоящий момент не действует.

AFTER – ***Поведение Бланка после выполнения действия***. После успешного выполнения действия можно, в случае, если вызывающая форма не закрылась в режиме CANCEL, текущий бланк поведёт себя в соответствии с указанными тут вариантами:

- NONE – Ничего не делать;
- FIELD –Текущее поле Бланка обновиться значением из свойства DEFINITION;
- CURRENT – Обновятся все поля той группы, к которой относится действие;
- ALL – Обновятся все поля Бланка;
- REFRESH – Переинициировать Бланк, то есть, как бы выйти и войти с теми же значениями входных параметров. Код из свойства ON_LOAD выполнится повторно;
- EXIT – Закрыть Бланк и передать управление в вызывающую форму. В случае визарда закроется весь визард;
- CANCEL – Отменить вызов Бланка. В вызывающей форме не будут выполняться POST-действия и перезапросы. В случае визарда откатится весь визард. Для отката только на предыдущий шаг визарда данное действие должно иметь ROLE = WIZARD.

ACTION_NOTES – ***Пояснения в Help***. Текст описания действия Бланка для отображения в Help и генерации документации. Следует помещать сюда всё, что не уместилось в одну фразу в свойстве EXPLANATION.

★ Свойства выходных параметров действий Бланка

Подчинённые действию элементы Бланка типа PARAM – это выходные параметры действия Бланка, передающие в вызываемую форму значения, которые становятся значениями её входных параметров.

В случае если набор выходных параметров действия не может быть заранее определён и становится известным только в момент выполнения действия, существует альтернативный способ передачи параметров: в виде коллекции, указанной в круглых скобках сразу после кода формы в свойстве COMMAND: `КОД_ФОРМЫ(Параметр1:Значение;Параметр2:Значение;...;)`. Возможен комбинированный вариант задания выходных параметров – и посредством подчинённых элементов действия типа PARAM и коллекци-

ей параметров в свойстве действия COMMAND. Параметры, определённые отдельными элементами, имеют приоритет.

NAME – ***Код параметра***. Реальный код выходного параметра. Именно по нему заполняется значение входного параметра в вызываемой форме. Если код пустой, то данный выходной параметр не работает (ничего в вызываемую форму передаваться не будет).

Выходной параметр с кодом CONFIG открывает вызываемую форму в указанной тут конфигурации. При отсутствии его в выходных параметрах, вызываемая форма открывается в той же конфигурации, что и вызывающая. Если в вызываемой форме нет нужной конфигурации, она открывается в конфигурации DEFAULT.

VALUE – ***Значение выходного параметра***. При использовании динамики её значение вычисляется непосредственно в момент выполнения действия. При совпадении по коду с входным параметром вызываемой формы его значение по умолчанию переопределяется данным значением выходного параметра.

DESCRIPTION – ***Комментарий***. Пояснения разработчика о значении выходного параметра.

★ Группы полей и действий Бланка

Поля и кнопки Бланка могут группироваться по смыслу. Сами группы, наряду с другими полями и кнопками, также могут включаться в группы более высокого порядка. Глубина вложенности теоретически не ограничена.

Свойства групп Бланка:

GROUP_TYPE – ***Тип группировки***. Поддерживаются следующие типы:

- TAB – Закладка или вкладка;
- BUNCH – Раскрывающаяся/свёрнутая группа;
- SIDE_BY_SIDE – Поля внутри группы будут располагаться горизонтально, друг за другом;
- Любое другое значение, включая пустую строку – Виртуальная группа, объединяющая элементы лишь общим заголовком.

Значение по умолчанию – пустая строка.

TITLE –***Заголовок группы***. Заголовок, объединяющий входящие в группу элементы. Для групп типа TAB – это надпись на закладке.

SEEABLE – ***Признак видимости***. Значения:

- Y (по умолчанию) – Группа видима
- Любое другое значение (включая пустую строку) – группа невидима и потому недоступны все вложенные в неё элементы;

POSITION – ***Положение подписей***. Расположение подписей к полям внутри группы относительно самих полей. Варианты:

- AUTO (по умолчанию) – решает движок, как считает более подходящим. В настоящий момент – слева от полей;
- LEFT – слева от всех полей;
- UP – сверху от всех полей.

Любое другое значение эквивалентно значению LEFT.

GROUP_NOTES – ***Пояснения в Help***. Текст описания группы элементов для отображения в Help и генерации документации.

★ Оперативные свойства Бланка и его элементов

- **{FormProperty:FormID}** – Уникальный в рамках сеанса текстовый идентификатор экземпляра формы. Не изменяется. Доступно в любом контексте.
- **{FormProperty:MetadataCode}** – Код метаданных, откуда загружался Бланк в рабочую область. Не изменяется. Доступно в любом контексте.
- **{FormProperty: MetadataApp}** – Идентифицирующий код приложения. Не изменяется. Доступно в свойствах Бланка.
- **{FormProperty:Code}** – Идентифицирующий код формы, определяющий её сущность. В частности, к этому коду привязываются пользовательские настройки. При загрузке

Бланка из метаданных свойство {FormProperty:Code} получается из общего свойства Бланка – FEATURE. Не изменяется. Доступно в свойствах Бланка.

- **{FormProperty:ConfigCode}** – Код конфигурации метаданных, из которой загружался Бланк. Не изменяется. Доступно в любом контексте.
- **{FormProperty:ConfigName}** – Наименование конфигурации метаданных, из которой загружался Бланк (в текущем языке). Не изменяется. Доступно в любом контексте.
- **{FormProperty:FType}** – Тип формы = BLANK. Не изменяется. Доступно в любом контексте.

- **{Property:Code}** – Код текущего элемента (в свойствах элемента). Не изменяется. Доступно в свойствах элементов.
- **{Property:EType}** – Тип текущего элемента: FIELD, ACTION, PARAM или GROUP. Не изменяется. Доступно в свойствах элементов.
- **{Property:Value}** – Текущее значение поля бланка. Доступно в свойствах полей и подчинённых им элементов (локальных действий и их параметров).
- **{Property:UserChanged}** – Y - Признак изменённости поля пользователем. При автоматическом перевычислении значения поля и только если значение изменилось, данный признак сбрасывается. Доступно в свойствах полей и подчинённых им элементов.
- **{Property:Diff}** – Y – Признак неопределённости значения поля. После изменения данных в поле это свойство сбрасывается. Доступно в свойствах полей и подчинённых им элементов.
- **{Property:Parent}** – Код родительского элемента. Для самостоятельных элементов тут значение «Form». При некоторых обстоятельствах может изменяться! Доступно в свойствах элементов

★ Системные переменные рабочей области Бланка

- **{Property:Var TransitParam}** – Коллекция значений параметров, указанных при загрузке метаданных в рабочую область. То есть это те параметры, с которыми открывалась форма, разработанная в метаданных.
- **{Property:Var CurrentField}** – Код текущего поля Бланка.
- **{Property:Var CurrentAction}** – Код текущего выполняющегося действия.
- **{Property:Var ParentFormID}** – ID рабочей области вызывающей формы.
- **{Property:Var Changed}** – Y – Признак наличия изменений в Бланке, сделанных пользователем.

Иерархическая структура на основе списков

Для построения иерархии используются специальные свойства уже существующих элементов в формах-списках. Иерархия выстраивается посредством действий Списков, вызывающих другие Списки. Таким образом, для отображения иерархической структуры достаточно при её вызове специальным способом открыть Список, все строки которого породят узлы верхнего уровня. Действия этого списка породят узлы второго уровня, а их раскрытие вызовет отображение Списка, вызываемого действием, в виде всё тех же узлов уже третьего порядка и т.д. Возможна укороченная схема иерархии, когда раскрытие узла-строки некоторого Списка отображает не действия над этой строкой, а сразу узлы-строки другого Списка.

Узлы, помимо текста, могут снабжаться пиктограммами и выделяться стилями. Существует понятие текущего узла, определяющего контекст – значения всех ячеек строки Списка, породившего этот узел. Пользователь может выделять несколько узлов на одном уровне (кликая мышкой с нажатым Ctrl).

Действия могут выполняться как над текущим узлом, так и над всеми выделенными узлами. После завершения действия может быть обновлён текущий узел или перечитан все узлы текущего уровня.

★ Особенности свойств уровня иерархии

Ниже описаны свойства Списка, участвующего в построении некоего уровня в иерархической структуре, в части, отличающейся от свойств Grid (см. «Общие свойства Списка» на стр.27)

QUERY_TEXT – ***SQL–выражение***. Основной SQL-запрос, формирующий множество элементов раскрываемого уровня иерархии. Множество значений столбцов станут контекстом порождаемого узла

HEADER – ***Текст родительского узла***. Он заменит текст раскрываемого узла. Если Список используется для порождения элементов самого верхнего уровня иерархии, HEADER становится заголовком окна формы-иерархии. Если это не первая форма в окне, то такой текст отображается как подзаголовок. Возможно, что один и тот же Список будет использоваться и для отображения Grid и для отображения Иерархии. В этом случае рекомендуется в каждой ситуации пользоваться разными конфигурациями Списка.

PORTION – ***Выбирать группами по***... Размер порции записей, отображающихся при раскрытии узла дерева. Если все порождаемые узлы не поместились в первую порцию, то последним станет узел специального типа с текстом в виде троеточия. Любое событие с этим узлом приведёт к отображению следующей порции узлов вместо этого специального узла.

NODE_TEXT – ***Текст узла дерева***. Текст подписи к узлу дерева. Обычно здесь используются ссылки на столбцы.

NODE_ICON – ***Иконка для каждого узла***. Имя иконки для отображения в узлах дерева.

ADD_INFO – ***Информация о записи***. Подсказка к каждому узлу порождаемого уровня иерархии. Обычно здесь используются ссылки на столбцы.

Свойство Списка START_MODE в режиме отображения иерархии не используется. Уровень иерархии всегда находится в многострочном режиме. Выделение нового узла без снятия уже существующих выделений производится кликом с нажатой клавишей Ctrl. Одновременно выделить можно узлы только одного уровня иерархии. Инверсию выделений включить нельзя.

★ Особенности свойств столбцов в уровне иерархии

Каждый столбец при формировании узла порождает значение контекста, ссылку на который (по коду столбца) можно использоваться в динамических свойствах. В частности, ссылки на значение элементов контекста необходимы для формирования текста узла уровня иерархии (см. свойство NODE_TEXT) и подсказок к ним (свойство ADD_INFO).

Свойства столбцов DATA_TYPE, FORMAT, CAPTION, ALIGNMENT, VISIBLE, WIDTH, VISUALIZATION, HINT, VARIANTS и TOTAL в режиме отображения иерархии не используются.

★ Особенности свойств действий в иерархии

Локальные действия (принадлежащие столбцам) в режиме отображения иерархии игнорируются. Остальные (глобальные) действия делятся на те, что порождают подчинённые узлы иерархии и те, что могут быть вызваны обычным образом, например, из локального меню (по правой кнопке мыши) или ролевыми клавишами.

Чтобы действие могло порождать подчинённые элементы иерархии, необходимы следующие условия:

- Действие должно быть глобальным (не подчинённым какому-то элементу);
- Свойство ACTIVE = EXIST;
- Свойство APPLY_TO = N;
- Свойство COMMAND_TYPE = LIST;
- Свойство PROC_INITIALLY должно быть пустым;
- Свойство COMMAND_TYPE = LIST;
- Свойство ROLE не должно иметь значения ADD, DELETE, CLOSE, AUTO, TIME и WIZARD;

При раскрытии узла иерархии отображаются подузлы с названиями (свойство PROMPT) всех действий, удовлетворяющих вышеперечисленным параметрам. А раскрытие уже узлов-действий повлечёт активацию нового уровня иерархии, основанного на указанном в действии Списке. Если же действие для раскрываемого узла иерархии только одно, то промежуточный уровень с названиями действий не образуется, а сразу раскрывается Список, вызываемый этим единственным действием (упрощённая схема).

Ниже описаны свойства действий Списка, участвующего в построении некоего уровня в иерархической структуре, в части, отличающейся от свойств действий в Grid (см. «Свойства действий Списка» на стр.32)

PROMPT – ***Наименование действия***. Заголовок действия при отображении в локальном меню или в качестве промежуточного уровня иерархии при раскрытии узла-строки Списка.

ACTIVE – ***Признак активности***. Надо понимать, что действия, для которых это свойство равно Y могут показываться, если текущим узлом является родительский узел уровня иерархии. Если же родительский узел в свёрнутом состоянии – это узел-строка вызывающего Списка (упрощённая схема), то для него набор

действий разный – в зависимости от его состояния: свёрнутый/развёрнутый. В свёрнутом состоянии – это строка родительского Списка и для неё действия порождаются родительским Списком. В развёрнутом состоянии – это заголовок дочернего (вызываемого) Списка и на него действия порождаются этим дочерним Списком.

ROLE – ***Способы активации***. Альтернативные способы активации действия:

- DBLCLICK – Двойной клик левой кнопкой мыши на узле или нажатие клавиши `Enter`;
- ADD – Нажатие клавиши `Insert`. По смыслу – действие добавляет строки на уровень иерархии;
- DELETE – Нажатие клавиши `Delete`. По смыслу – действие удаляет строки с уровня иерархии;
- CLOSE – Закрытие формы. Срабатывает только для иерархии самого верхнего уровня;
- AUTO – Действие, выполняющееся при каждом изменении значения любого его выходного параметра;
- ALLEXPAND – Автораскрытие узла-действия (и всех вложенных узлов).

AFTER – ***Поведение уровня после выполнения действия***. После успешного выполнения действия текущий уровень иерархии поведёт себя в соответствии с указанными тут вариантами:

- NONE – Ничего не произойдёт;
- FIELD – Ничего не произойдёт;
- CURRENT – Обновится текущий узел;
- ALL – Обновятся все узлы текущего уровня. Текущий узел попытается сохраниться (без гарантии);
- EXIT – Свернуть текущий уровень иерархии или закрыть иерархическую структуру для иерархии самого верхнего уровня;
- CANCEL – Отменить вызов формы – иерархической структуры. Срабатывает только для иерархии самого верхнего уровня.

Свойства ICON и BUTTON игнорируются.

У параметров действий в иерархических структурах никаких особенностей – по сравнению со Списками – нет.

★ Оперативные свойства иерархической структуры

- **{Property:FormID}** – Уникальный в рамках сеанса текстовый идентификатор рабочей области, куда загружен Список для отображения текущего уровня иерархии. Не изменяется. Доступно в любом контексте.
- **{Property:MetadataCode}** – Код метаданных, откуда загружался Список в рабочую область. Не изменяется. Доступно в любом контексте.
- **{Property: MetadataApp}** – Идентифицирующий код приложения. Не изменяется. Доступно в свойствах Списка.
- **{Property:ConfigCode}** – Код конфигурации метаданных из которой загружался Список. Не изменяется. Доступно в любом контексте.
- **{Property:ConfigName}** – Наименование конфигурации метаданных из которой загружался Список (в текущем языке). «Виртуальная конфигурация», если Список чисто программный. Не изменяется. Доступно в любом контексте.
- **{Property:FType}** – Тип формы = TREE. Не изменяется. Доступно в любом контексте.
- **{Property:SelectionMode}** – Режим выделения строк в Списке: Single или Multi. Доступно в любом контексте.
- **{Property: CountSelected}** – Количество выделенных узлов в уровне иерархии.
- **{Property:EType}** – Тип текущего элемента (в свойствах элемента): ACTION или PARAM. Не изменяется. Доступно в свойствах элементов.
- **{Property:Parent}** – Код действия для выходного параметра. Для действий тут всегда значение «Form». Доступно в свойствах элементов

★ Системные переменные рабочей области иерархической структуры

- **{Property:Var TransitParam}** – Коллекция значений параметров, указанных при загрузке метаданных в рабочую область. То есть это те параметры, с которыми открывалась форма, разработанная в метаданных.

- **{Property:Var CurrentRow}** – Номер по порядку текущего узла в уровне. Пустое значение – текущая строка сейчас не в текущем уровне.
- **{Property:Var FetchedRows}** – Количество извлечённых узлов в уровне иерархии на текущий момент.
- **{Property:Var IsLastPage}** – Признак полноты полученных узлов в уровне иерархии: Y – все узлы уже получены. Любое другое значение – на сервере, возможно, есть ещё узлы, не полученные в данном уровне иерархии.
- **{Property:Var MainQuery}** – Текст основного запроса уровня иерархии
- **{Property:Var CurrentAction}** – Код текущего выполняющегося действия.
- **{Property:Var ParentFormID}** – ID рабочей области вызывающей формы.
- **{Property:Var ParentNode}** – Уникальный идентификатор родительского узла.
- **{Property:Var CurrentNodeType}** – Тип текущего узла в уровне:
  ROW – узел-строка Списка
  ACTION – узел-действие
  CONTINUE – специальный узел для запроса следующей порции узлов-строк.
- **{Property:Var RootFormID}** – ID рабочей области высшего уровня иерархии.
- **{Property:Var ContextForkID}** – ID рабочей области текущего уровня иерархии (где сейчас текущая строка или выделены несколько строк). Это значение есть только у рабочей области высшего уровня иерархии.

Команда ОС

Сервер приложений LUI может выполнять команды операционной системы (ОС) по требованию прикладной системы. Выполнение команд производится в той операционной системе, в которой запущен сервер приложений. Выполнение команд производится от имени и в правах того пользователя операционной системы, от имени которого запущен сервер приложений. Целями выполнения команд ОС для прикладных систем могут быть: выполнение SQL-скриптов, формирование отчётов, интеграция со смежными системами и т.д. Для того чтобы прикладная система могла выполнять команды ОС, в ней должны быть созданы соответствующие формы команд ОС. В отличие от меню, списков, бланков формы команд ОС не являются экранными. Тип таких форм имеет мнемонический код “HOST”. Как и всякая форма, форма команды ОС создаётся редактором форм в одной из групп форм.

★ Общие свойства формы команды ОС

Code – ***Уникальный в рамках прикладной системы код формы***. Этот код вводится при создании формы команды ОС и не меняется в разных её конфигурациях. По этому коду в совокупности с кодом конфигурации выполняется идентификация и загрузка формы команды ОС из базы данных для выполнения в прикладной системе. Этот код указывается в действиях форм, которые вызывают команду ОС. Уникальный код может быть изменён в редакторе форм LUI. При этом автоматических изменений кода в действиях вызывающих форм не выполняется. Это надо делать вручную. Рекомендуется, чтобы код состоял из:

- Краткого обозначения прикладной системы
- Суффикса “H”
- Мнемоники, отражающей смысл команды

Пример: LUI_H_PSQL – вызов утилиты “psql”.

COMMAND_VARIABLE_FORMAT – ***Формат переменной-параметра***. Довольно часто бывает такая ситуация, когда разработка команды ОС выполняется в условиях неизвестности всех параметров, которые будут переданы в разных случаях выполнения команды. Примером тому является разработка команды ОС, которая должна выполнять самые разные sql-скрипты. При этом каждый sql-скрипт может использовать собственный набор входных параметров, не похожий на набор параметров другого скрипта. Для таких случаев в LUI предусмотрена обработка параметров, не описанных явно в числе входных параметров формы команды ОС. Такие параметры называются параметры-переменные. Каждый параметр-переменная являются носителем двух свойств: “Code” и “Value”. Различные программы, выполняемые в среде ОС сервера приложений, могут иметь различные требования по представлению таких параметров в командной строке. Так, например, утилита psql – (Интерактивный терминал Postgres) пропускает параметры внутрь выполняемого sql-скрипта, если каждый из них представлен в следующем формате: -v Код_Параметра=”Значение_Параметра”. Таким образом, при формировании текста команды каждый такой параметр должен быть подвергнут форматированию в соответствии с требованиями вызываемой програм-

мы. Свойство COMMAND_VARIABLE_FORMAT задаёт выражение для вычисления отформатированного представления параметра-переменной.

Для psql следует ввести

-v {Property:Code}="{Property:Value}"

В процессе формирования текста команды сервер приложений вычислит дополнительное свойство команды ОС, которое будет содержать полный перечень отформатированных представлений параметров-переменных, отделённых друг от друга пробелом. Это дополнительное свойство будет иметь код “Variables”. Ссылку на это свойство в виде {Property:Variables} можно использовать в выражении вычисления текста команды – COMMAND_TEXT.

COMMAND_TEXT – *Текст команды*. Это свойство должно содержать выражение, которое вычисляет полный текст команды ОС. При написании выражения можно ссылаться на входные параметры, свойство Variables, на любые свойства приложения (глобальные параметры).

Пример выражения для psql:

psql -f "{F:P_SCRIPT}" {Property:Variables} –o “{ApplicationProperty:TmpDir}{F:P_OUTPUT}”

{JS:"{F:P_DBURL}".replace("jdbc:","").replace("//","//{F:P_USERNAME}:{DECR:{F:P_PASSWORD}}@")}

Пусть P_SCRIPT=C:\mydir\script.sql

ApplicationProperty:TmpDir= T:\\lui2\LUI\lui\11394\

P_OUTPUT=output.htm

P_DBURL= jdbc:postgresql://localhost:5432/postgres

P_USERNAME=testuser

P_PASSWORD=*****

То результирующий текст будет такой:

psql –f “C:\mydir\script.sql” –o “T:\\lui2\LUI\lui\11394\output.htm” postgresql://testuser:testuser@localhost:5432/postgres

COMMAND_WRKDIR– *Рабочий каталог*. Это свойство определяет текущий рабочий каталог при выполнении команды ОС. Обычно им должен быть временный каталог сеанса пользователя приложения –

{ApplicationProperty:TmpDir}

Установка текущего рабочего каталога полезна в случае вызова программы, которая выводит некие результаты, журнал, ошибки и т.п. в файлы без указания полного пути.

COMMAND_RESULT– *Результат команды*. Это свойство определяет выражение, которое вычисляет результат выполнения команды. Обычно это код возврата, полученный от операционной системы, - {Property:RESULT_CODE}

Однако иногда может потребоваться вернуть в вызывающую программу не код возврата, а, например, имя сгенерированного файла в форме гиперссылки с целью отобразить его содержимое в окне браузера. В этом случае следует использовать

{ApplicationProperty:TmpDirURL}{F:P_OUTPUT}

★ Свойства входных параметров команды ОС

Форма типа HOST может иметь элементы только типа “PARAM”-входной параметр. Эти параметры используются для динамического формирования полного текста команды, который будет передан операционной системе для выполнения. При выполнении значения параметров передаются команде ОС из вызывающей формы. Если вызывающая форма не передаёт значение параметра (параметр отсутствует при вызове), то будет действовать значение параметра по умолчанию.

Code – ***Код входного параметра.*** Задаётся при создании параметров. Именно по этому коду производится связывание выходных параметров действия в вызывающей форме с входными параметрами меню. Код параметра рекомендуется снабжать префиксом “P_”. Пример: P_CODE. ». Следует заметить, что в отли-

чие от столбцов и полей списков и бланков параметры всегда неявно обладают типом данных CHAR. Поэтому при кодировании свойств форм запись '**{F:КОД}**' всегда эквивалентна **{F2SQL:КОД}**.

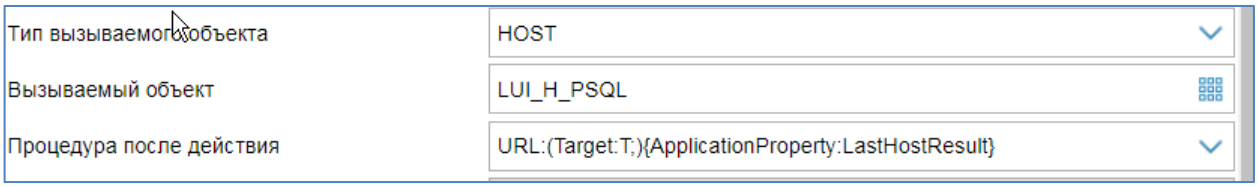
Name – ***Название входного параметра.*** Задаётся при создании параметров. Используется для отображения в навигаторе редактора форм. Если приложение многоязычное, то название рекомендуется задавать на всех языках приложения.

VALUE – ***Значение по умолчанию***. Если вызывающая Список форма не укажет значение параметра, то при ссылке на данный входной параметр в динамических свойствах будет подставляться указанное тут значение. По умолчанию – пустая строка. Значение по умолчанию может содержать динамические элементы. Если вызывающая форма переопределила значение по умолчанию входного параметра, то указываемая здесь динамика вычисляться никогда не будет.

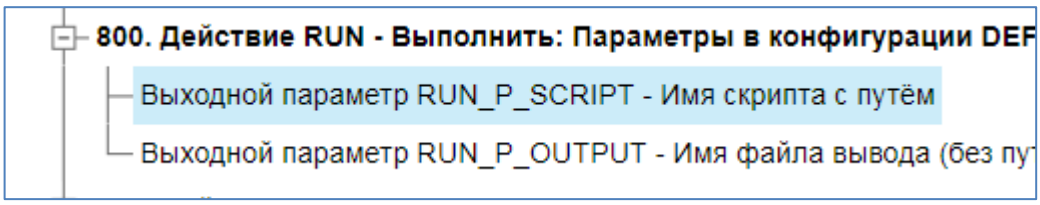
DESCRIPTION – ***Описание***. Комментарий разработчика о входном параметре и о том, на что он влияет.

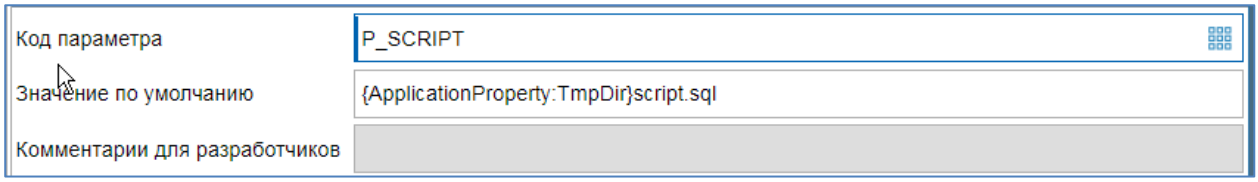
★ Вызов команд ОС с помощью действий

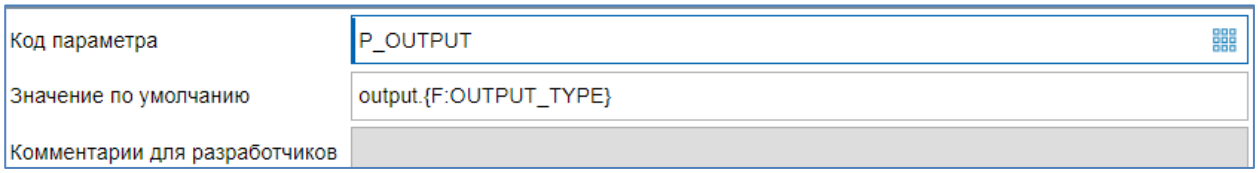
Формы команд ОС не являются экранными. При их вызове не возникает каких-либо специальных списков и бланков. Тем не менее, формы команд ОС вызываются так же, как и экранные формы. В вызывающей форме действие, которое вызывает команду ОС, может быть оформлено, например, в следующем виде:



Выходные параметры:







Следует иметь в виду, что результат выполнения команды ОС всякий раз записывается в глобальный параметр приложения LastHostResult и доступен через {ApplicationProperty: LastHostResult}.

★ Вызов команд ОС оператором метаязыка LUI

Помимо этого существуют ещё два способа вызова команд ОС, реализованных в стандартном подключаемом модуле LUI.

Первый способ – это вызов компоненты Execute. Этот вызов может быть написан в свойстве “Процедура перед действием” или в свойстве “Процедура после действия” любого действия любой формы.

Синтаксис вызова:

HOST:***Команда***;***Параметр1***:***Значение1***;…***ПараметрN***:***ЗначениеN***;

Второй способ – это вызов компоненты Compute для вычисления результата выполнения команды ОС.

Синтаксис вызова:

{**HOST**:***Команда***;***Параметр1***:***Значение1***;…***ПараметрN***:***ЗначениеN***;}

Это выражение вернёт результат выполнения команды ОС (COMMAND_RESULT).

Для вызова команды ОС нужной конфигурации следует использовать параметр CONFIG.

Задача

Задача это некая рутинная программа, которая может вызываться действиями из экранных форм или выполняться в фоновом режиме по расписанию. Задача может не только выполнять операции в БД, но и в файловой системе сервера приложений, в смежных системах и прочие действия необходимые прикладной системе. Например – массовая отправка e-mail-оповещений, обнаружение файла и загрузка данных из него в БД. Для того чтобы прикладная система могла выполнять задачи, в ней должны быть созданы соответствующие формы задач. В общем случае задача состоит из трёх основных компонентов:

- Запрос к источнику данных (это либо оператор select языка SQL, либо обращение к строкам текстового файла)
- Действие, выполняемое перед выполнением запроса к данным
- Действие, выполняемое для каждой извлечённой строки
- Действие, выполняемое после обработки всех строк

★ Общие свойства формы задачи

В отличие от меню, списков, бланков формы задач не предназначены для ввода и отображения данных пользователем. Тип таких форм имеет мнемонический код “TASK”. Как и всякая форма, форма задачи создаётся редактором форм в одной из групп форм.

Code – ***Уникальный в рамках прикладной системы код формы***. Этот код вводится при создании формы задачи и не меняется в разных её конфигурациях. По этому коду в совокупности с кодом конфигурации выполняется идентификация и загрузка формы задачи из базы данных для выполнения в прикладной системе. Этот код указывается в действиях форм, которые вызывают задачу. Уникальный код может быть изменён в редакторе форм LUI. При этом, автоматических изменений кода в действиях вызывающих форм не выполняется. Это надо делать вручную. Рекомендуется, чтобы код состоял из:

- Краткого обозначения прикладной системы
- Суффикса “T”
- Мнемоники отражающей смысл команды

Пример: LUI_T_DELSES – удаление данных об устаревших сеансах пользователей .

TASK_PRG_BEGIN – ***Действие в начале***. В этом свойстве можно ввести программу, которая будет выполняться в самом начале каждого запуска задачи. Например:

```
SendMail:Запуск задачи;
Content:Задача {Property:Code} запущена в {SQL:select current_timestamp};
Host:my_smtp_server;
Port:25;
Sender:Lui@myorg.ru;
Recipients:admin@myorg.ru;
```

TASK_QUERY – *Действие в начале*. Это свойство должно содержать либо запрос на языке SQL, либо обращение к файлу на языке LUI. Запрос кодируется, например, так:

```
select * from
 lui_t_session
where start_datetime<CURRENT_TIMESTAMP-{F2SQL:P_BEGIN_DAYS_AGO}
order by id
```

Здесь {F2SQL:P_BEGIN_DAYS_AGO} – ссылка на значение входного параметра, которое преобразовано к форме, понятной для СУБД.

Обращение к файлу кодируется так:

```
FILE:<имя файла с путём>;CHARSET:<кодировка>;
```

Пример:

```
FILE:c:\temp\my_file.txt;CHARSET:windows-1251;
```

Следует заметить, что свойство TASK_QUERY может быть не заполнено. Это означает, что вся функциональность задачи закодирована в свойствах TASK_PRG_BEGIN и/или TASK_PRG_END.

TASK_PRG_ROW – *Действие для каждой строки*. Это свойство определяет программу на метаязыке LUI, которая будет выполняться для каждой извлечённой строки запроса или файла.

При кодировании этого свойства следует использовать ссылки на столбцы запроса:

```
{QC:<имя столбца>}
```

или на строку файла:

```
{FormProperty:FileRecord}.
```

Номер текущей строки запроса или файла доступен в переменной {FormProperty:ProcessingRowNumber}.

Пример обработки строки запроса:

```
SQL:delete from lui_t_session where id={QC:id}


SQL:commit


HOST:LUI_H_DEL_DIR;P_DIRNAME:{ApplicationProperty:APP_FILES_PATH}{QC:app_code}
{ApplicationProperty:PathSeparator} {QC:user_name} {ApplicationProper-
ty:PathSeparator}{QC:id}{ApplicationProperty:PathSeparator}
```

Эта программа состоит из трёх инструкций. Первая удаляет данные в таблице. Вторая фиксирует транзакцию. Третья удаляет каталог вместе с содержимым.

Пример обработки строки файла:

```
SQL:insert into test_table(c_text) values ('{FormProperty:FileRecord}')
```

Эта программа делает вставку в таблицу всей строки считанной из файла.

TASK_PRG_ROW_EX – *Обработка ошибок для строки*. Это свойство определяет программу на языке LUI, которая будет выполняться в случае возникновения ошибки в программе обработки строки. Если это свойство пусто, то в случае возникновении ошибки задача будет аварийно завершена и получит состояние DISABLE. Перед завершением будет выполнена программа из TASK_PRG_EXCEPTION

Если это свойство содержит программу реакции на ошибку обработки строки, то при возникновении ошибки будет выполнена эта программа и задача продолжит свою работу.

Если ошибка возникнет в самой программа обработки ошибки, то задача будет аварийно завершена и получит состояние DISABLE.

TASK_PRG_END – *Действие в конце*. В этом свойстве можно ввести программу, которая будет выполняться в самом конце каждого запуска задачи после обработки всех строк запроса или файла.

Например:

```
SendMail:Завершение задачи;
Content:Обработано строк - {FormProperty:ProcessingRowNumber};
Host:my_smtp_server;
Port:25;
Sender:Lui@myorg.ru;
Recipients:admin@myorg.ru;
```

TASK_PRG_EXCEPTION – *Действие при завершении по ошибке*. В этом свойстве можно ввести программу, которая будет выполняться в случае аварийного завершения задачи при возникновении не обработанных исключений.

Текст сообщения об ошибке доступен в переменной {FormProperty:TASK_ERROR_TEXT}

Пример:

```
SendMail:Авария задачи {Property:Code};
Content{FormProperty:TASK_ERROR_TEXT};
Host:my_smtp_server;
Port:25;
Sender:Lui@myorg.ru;
Recipients:admin@myorg.ru;
```

TASK_BAR_PCT_TEXT и TASK_BAR_WRK_TEXT – *Текст на прогресс-индикаторе и Текст над прогресс-индикатором*.

В ходе работы задачи интерфейс пользователя отображает прогресс-индикатор. Эти два свойства позволяют задать текст, отображаемый на в прогресс-индикаторе и над прогресс-индикатором соответственно.

В первом обычно кодируются %% выполненной работы:

```
{FormProperty:ProcessingPercent}%
```

или отношение номера текущей строки запроса к общему количеству строк:

```
{FormProperty:ProcessingRowNumber}/{FormProperty:TotalRowNumber}
```

Во втором принято отображать информацию о времени работы:

```
{FormProperty:ProcessingElapsedTime}
```

и/или о содержании обрабатываемой строки:

```
обработка строки ID={QC:id}
```

TASK_SHOW_PARAM_FORM – *Отображать форму параметров*. Это свойство позволяет включить (Y) или выключить (N) отображение формы параметров при запуске задачи в синхронном режиме. Если отображение отключено, то форма ввода параметров не возникает и задача запускается немедленно при вызове. При этом значения всех параметров должны передаваться из вызывающей формы, иначе они получат значения по умолчанию, установленные разработчиком задачи.

HEADER – *Заголовок*. В этом свойстве задаётся заголовок формы ввода параметров при запуске задачи в синхронном режиме.

★ Свойства входных параметров Задачи

Форма типа TASK подобно прочим формам может иметь входные параметры. Однако, это единственный тип элемента который может иметь задача. В отличие от входных параметров других форм, параметры задачи могут отображаться в виде полей в специальном бланке при запуске задачи в синхронном режиме и

при настройке асинхронных задач, выполняемых в фоновом режиме по расписанию. Поэтому параметры обладают набором свойств, который полностью совпадает с набором свойств поля бланка.

Тип элемента “входной параметр задачи” имеет мнемонический код “ITEM”, как у полей бланка.

Параметры могут использоваться для модификации поведения задачи при её применении в различных случаях. При выполнении задачи в синхронном режиме значения параметров передаются из вызывающей формы. Если значение параметра не передано, то будет действовать значение, заданное в свойстве DEFINITION- “Значение (выражение) по умолчанию”.

При разработке задачи, параметры создаются следующим образом. Необходимо раскрыть узел “Элементы задачи”, нажать правую кнопку мыши и выбрать действие “Добавить”. В бланке “Создание нового элемента формы” укажите тип элемента – “Входной параметр”, код элемента и название.

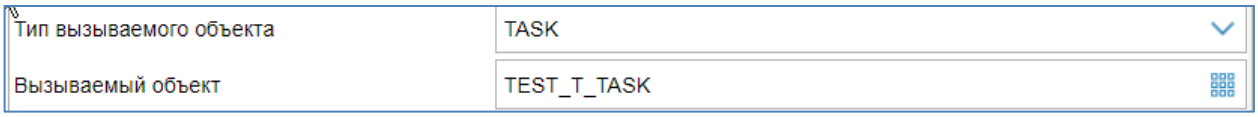
Название используется как информация для разработчиков.

Код параметра рекомендуется снабжать префиксом “P_”.

★ Синхронное выполнение Задачи

В синхронном режиме задача запускается из вызывающей формы. Вызывающая форма при этом становится не активной и не получает управления до нормального или аварийного завершения задачи.

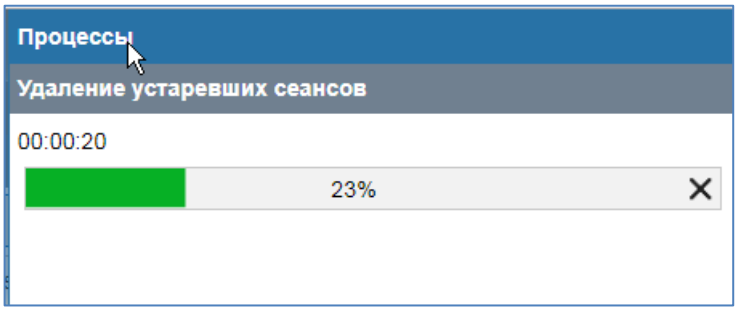
Для запуска задачи в синхронном режиме вызывающая форма должна иметь соответствующее действие. В котором свойства COMMAND_TYPE и COMMAND заполнены, например, так:



Параметры задачи кодируются в выходных параметрах этого действия.

В ходе синхронного выполнения задачи в интерфейсе пользователя отображается прогресс-индикатор.

Пример:



Справа от индикатора имеется иконка “X”. Нажатие на неё приведёт к прерыванию выполнения задачи. Следует иметь в виду, что реакция на нажатие этой иконки будет только после завершения работающей программы обработки очередной строки.

★ Асинхронное выполнение Задач

Настройка асинхронного выполнения Задач

Для выполнения задач в асинхронном фоновом режиме сервер приложений LUI в конфигурационном файле должен иметь не нулевое значение параметра taskDispatcherTimeOut.

Этот параметр устанавливает периодичность в миллисекундах опроса очереди желающих выполниться задач. Рекомендуемое значение:

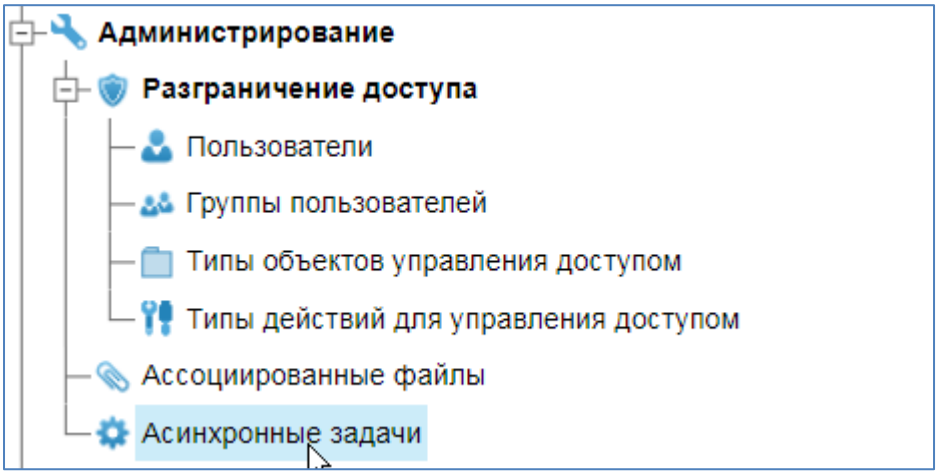
taskDispatcherTimeOut=3000

Это соответствует частоте опроса 1 раз каждые 3 секунды.

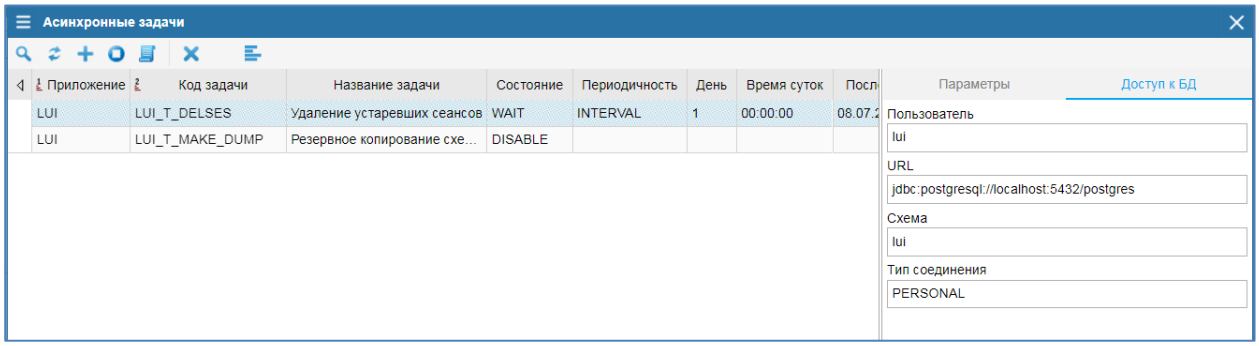
Асинхронное выполнение задачи происходит в соответствии с заданным для неё расписанием. Когда приближается её время выполнения (т.е. до старта остаётся три или менее секунд), сервер приложений за-

пускает процесс выполнения задачи. Процесс выполнения задачи дожидается требуемого времени старта и выполняет задачу.

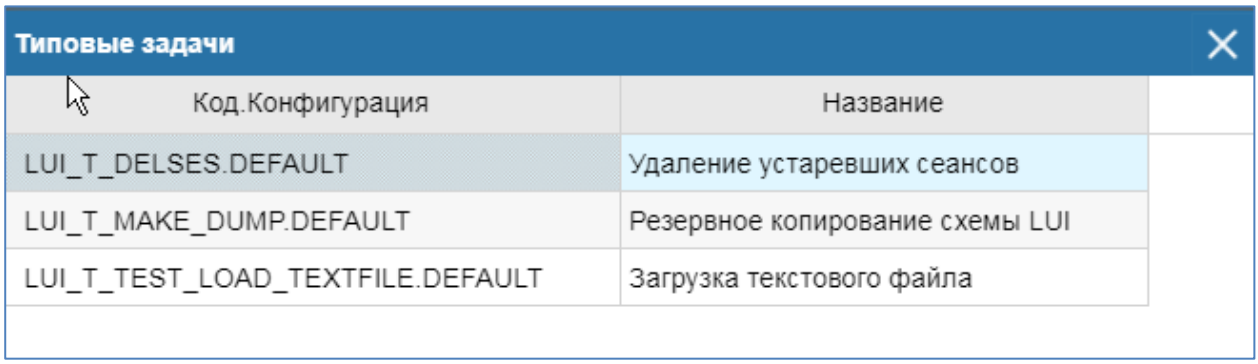
Для работы задачи в асинхронном фоновом режиме её требуется зарегистрировать в списке асинхронных задач. Список асинхронных задач доступен из главного меню прикладной системы созданной в LUI. Таким образом, роль настройки асинхронных выполнений задач возложена не на разработчика прикладной системы, а на её пользователя – администратора. Пример пункта главного меню:



Список асинхронных задач выглядит так:



Для регистрации задачи следует выполнить действие “Добавить” (иконка) и выбрать форму и конфигурацию одной из задач, хранящихся в метаданных. Пример списка выбора:



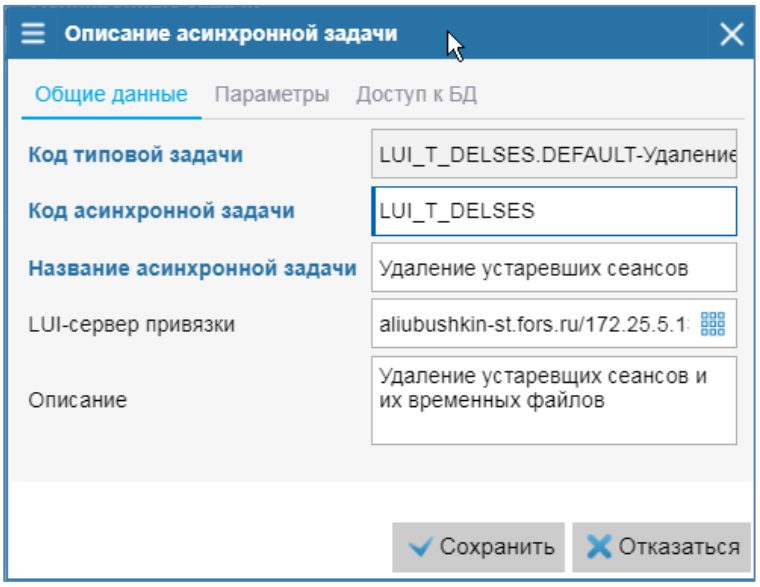
Для одной и той же конфигурации можно создать две и более асинхронные задачи с разными расписаниями и параметрами выполнения.

После создания задача получает статус DISABLE – отключена.

Для настройки основных свойств задачи следует выполнить действие “Изменить” (Иконка). В бланке редактирования свойств задачи на закладке “Общие данные” можно:

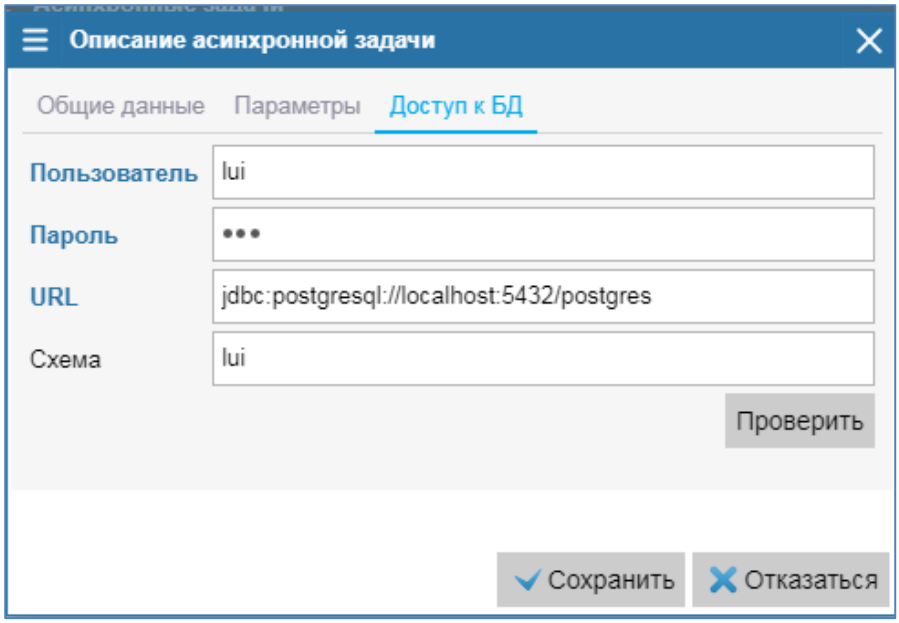
- Увидеть код и конфигурацию типовой задачи в метаданных

- Изменить уникальный код асинхронной задачи (по умолчанию он синтезируется из кода задачи хранящейся в метаданных)
- Изменить название задачи
- Указать сервер приложений, на котором, и исключительно на котором, должна выполняться задача в кластерных инсталляциях LUI
- Изменить описание.



На закладке “параметры” следует установить значения параметров задачи.

На закладке “доступ к БД” следует установить параметры соединения задачи с прикладной БД.



Запуск

Для запуска однократного или периодического выполнения задачи следует выполнить для неё действие “Запустить” (иконка).

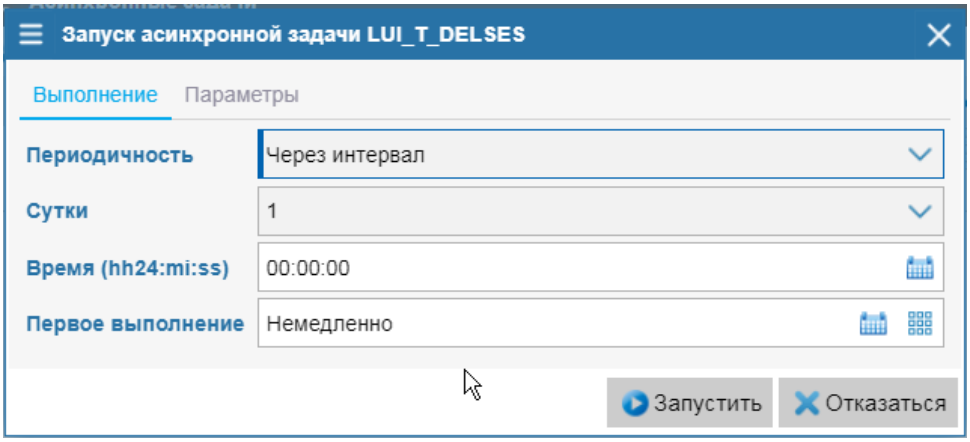
В бланке запуска на закладке “Выполнение” следует указать периодичность и дату/время первого выполнения.

Поле “Периодичность” может принимать следующие значения:

- “Однократно” (ONCE): это однократное выполнение задачи в указанные дату и время

- “Через интервал” (INTERVAL): это периодическое выполнение задачи через указанный интервал времени. Задача будет запускаться с указанной периодичностью. Однако если очередное выполнение задачи затянется и превысит дату и время следующего старта, то этот следующий старт будет отложен до момента завершения работы задачи. Таким образом, если время работы задачи будет всегда превышать указанный интервал, задача будет запускаться с интервалом равным продолжительности её работы.
- “Через паузу” (PAUSE): это многократное выполнение задачи, когда после каждого успешного её завершения делается пауза указанной длительности
- “Ежедневно” (DAILY): это ежесуточное выполнение задачи в указанное время суток
- “Еженедельно” (WEEKLY): это еженедельное выполнение задачи в указанный день недели и в указанное время суток
- “Ежемесячно” (MONTHLY): это ежемесячное выполнение задачи в указанный день месяца и в указанное время суток, при этом день месяца задаётся числом от 1 до 28 или одним из значений: “Последний”, “Предпоследний”, ”Перед предпоследним”.

Пример:



На закладке “Параметры” имеется возможность изменить значения параметров задачи непосредственно перед её запуском.

После запуска задача получает статус “WAIT” – ожидание наступления времени выполнения.

Когда приблизится требуемое время выполнения, диспетчер задач сервера приложений запустит процесс её выполнения и переведёт задачу в состояние “PREPARE” – подготовка к выполнению.

Для наблюдения за изменением состояний можно в списке задач выполнять действие “Выполнение запроса” (F8).

Когда наступит время выполнения задачи, процесс её выполнения начнёт её выполнять и переведёт задачу в состояние “RUN”.

После нормального завершения работы, если задача запущена для однократного выполнения, то она перейдёт в состояние “DISABLE” – отключена, если задача запущена для многократного выполнения, то она перейдёт в состояние “WAIT”.

В случае аварийного завершения задача всегда переходит в состояние “DISABLE”.

При этом код задачи отображается красным.

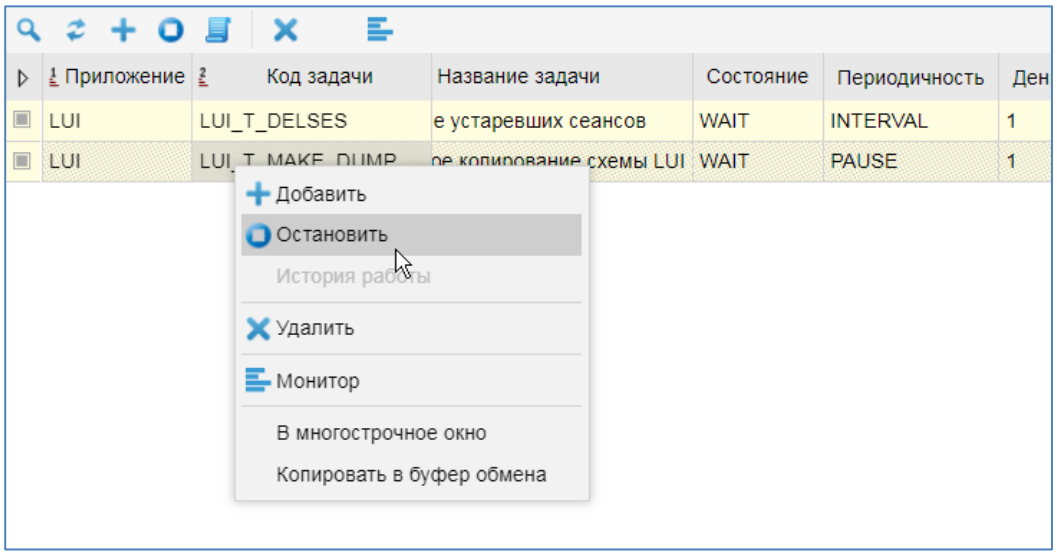
Следует иметь в виду, что запланированное выполнение асинхронной задачи будет произведено независимо от наличия или отсутствия сеансов пользователей приложения, для которого задача создана. При работе задачи создаётся самостоятельный отдельный сеанс работы в приложении. Этот сеанс создаётся от имени того пользователя приложения, который выполнил запуск задачи.

Останов

Периодическое выполнение задачи может быть остановлено, и даже текущее её выполнение может быть прервано. Это делается действием “Остановить” в меню (возникает по правой кнопке мыши) или иконкой в панели инструментов.

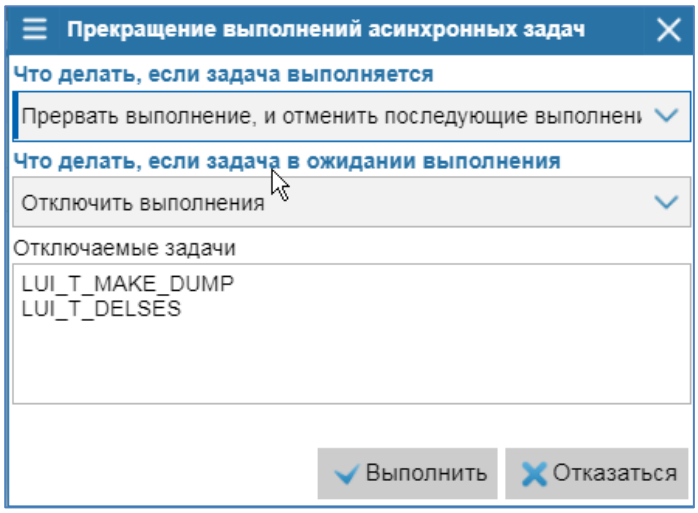
Это действие можно выполнять над несколькими выделенными задачами.

Пример:



При выполнении этого действия возникает бланк для указания требуемых действий.

Пример:



Поле “Что делать, если задача выполняется” может иметь следующие значения:

- BREAK: Прервать выполнение, но продолжать последующие выполнения;
- ABORT: Прервать выполнение, и отменить последующие выполнения;
- STOP: Дождаться завершения и отменить последующие выполнения;
- NONE: Ничего (задача не будет затронута данной операцией);

Значение этого поля управляет действиями в отношении задач находящихся в состоянии “RUN” или “PREPARE”.

Поле “Что делать, если задача в ожидании выполнения” может иметь следующие значения:

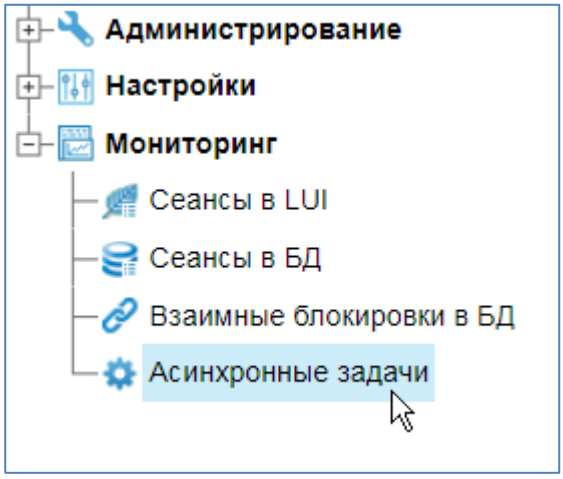
- STOP: Отключить выполнения;
- NONE: Ничего (задача не будет затронута данной операцией);

Значение этого поля управляет действиями в отношении задач находящихся в состоянии “WAIT”.

Задачи, находящиеся в состоянии “DISABLE” к моменту выполнения над ними этого действия, не будут затронуты, даже если они были среди выделенных задач.

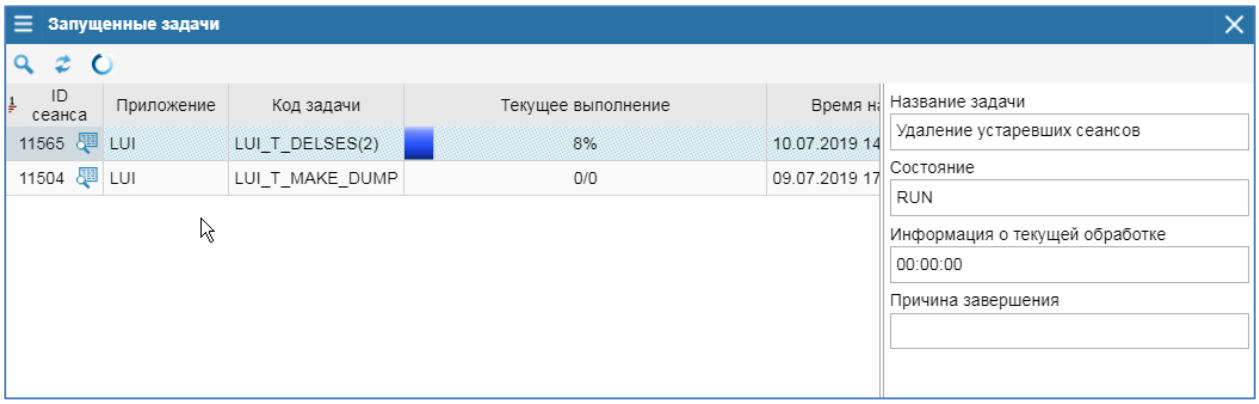
Мониторинг

Для наблюдения за изменением состояний задач и ходом их выполнения существует соответствующий пункт главного меню:



При выборе этого пункта меню отображается список асинхронных задач не находящихся в состоянии DISABLE, т.е. список запущенных задач. Если этот список открыт в прикладной системе LUI (т.е. в среде разработки приложений) то список содержит запущенные задачи всех приложений. Если этот список открыт в иной прикладной системе, то список будет содержать запущенные задачи данной прикладной системы.

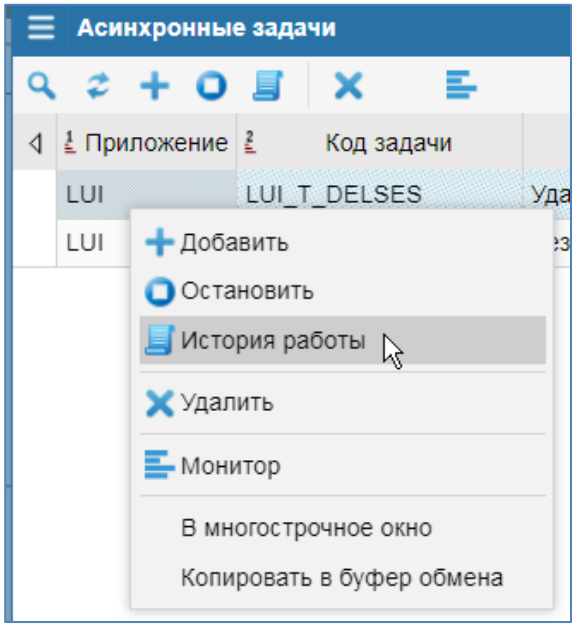
Если задача находится в стадии выполнения, то для неё в столбце “Текущее выполнение” отображается прогресс-индикатор.



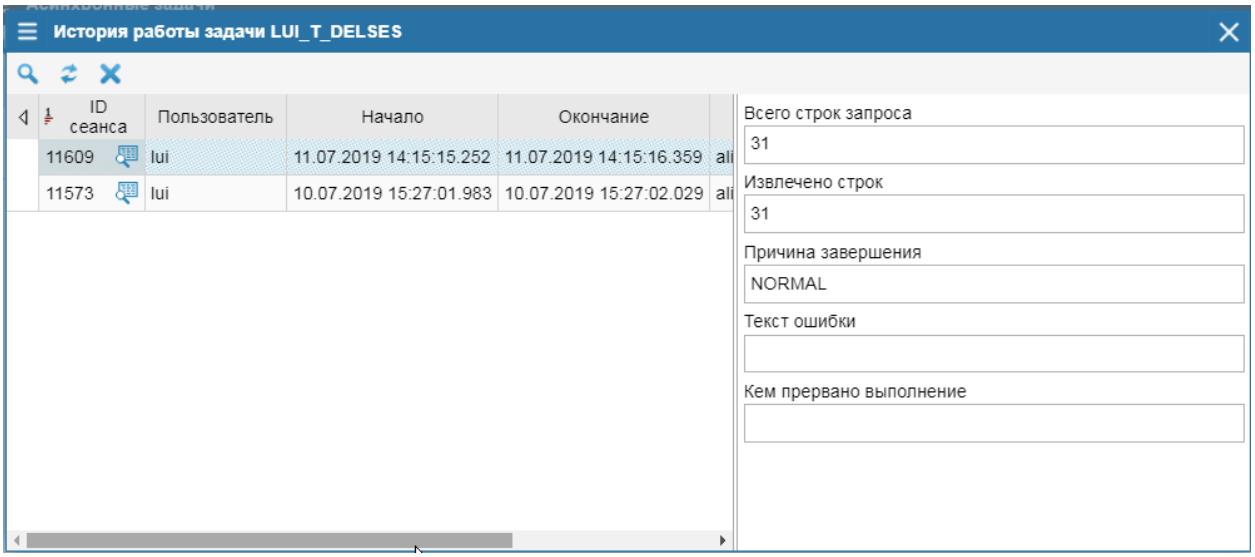
По умолчанию этот список открывается в режиме постоянного (1 раз в секунду) перезапроса. Отключение этого режима делается нажатием на иконку .

История работы задачи

В списке асинхронных задач для задачи можно применить действие “История работы”.

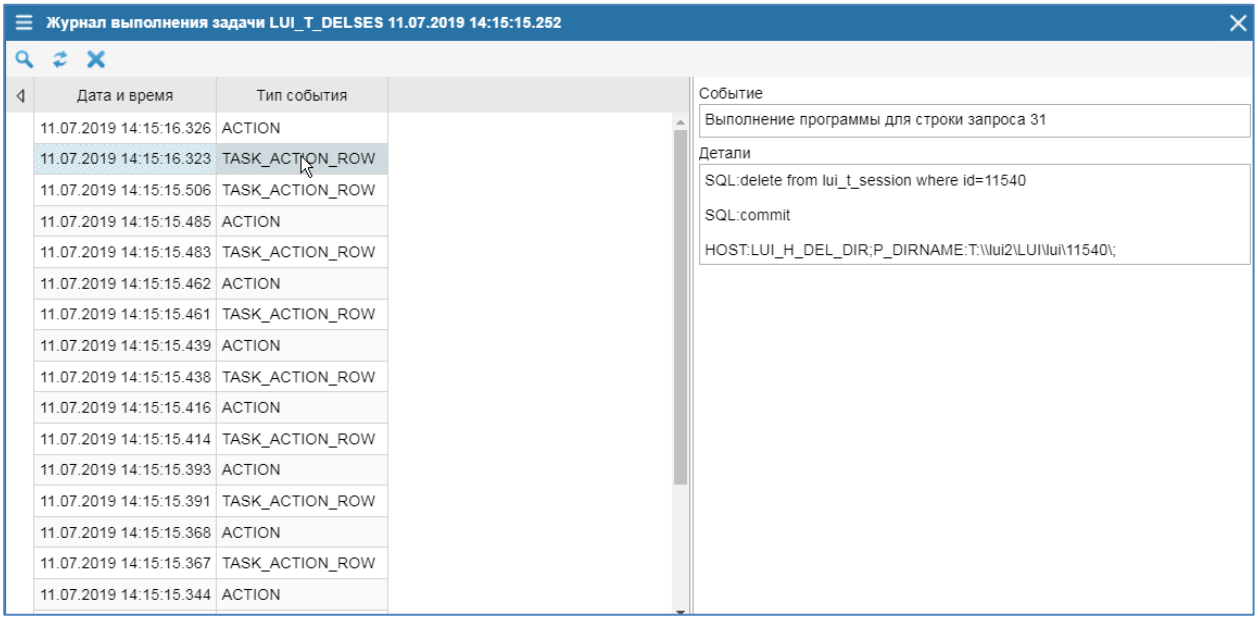


По этому действию отображается список сеансов инициированных задачей в прикладной системе.



Журнал выполнения задачи

В списке “История работы задачи” и в списке “Запущенные задачи” имеется действие “Журнал событий” (иконка). По этому действию вызывается список “Журнал выполнения задачи”. В нём отображаются действия выполненные задачей в течение сеанса. Журналированию подлежат не все действия, выполняемые для каждой строки запроса или файла, а только первые 10 действий и последнее действие.



★ Системные переменные рабочей области Задачи

При кодировании этих свойств можно использовать следующие свойства приложения и формы:

{ApplicationProperty:AppFilePath} – путь к файлам приложения;

{ApplicationProperty:TmpDir} – путь к каталогу для временного хранения файлов в течении сеанса в приложении;

{ApplicationProperty:AppUser} – имя пользователя, от имени которого запущена задача (в синхронном режиме это пользователь, который вошёл в приложение, в асинхронном – это пользователь, который запустил задачу для многократного или однократного выполнения);

{ApplicationProperty:SessionID} – id сеанса в приложении (инициирован пользователем приложения при выполнении задачи в синхронном режиме, инициирован диспетчером задач при выполнении в асинхронном режиме);

{ApplicationProperty:PathSeparator} – разделитель между именами каталогов (“\” или “/”);

{ApplicationProperty:dbUser} – пользователь, указанный при подключении к БД;

{ApplicationProperty:dbSchema} – схема, указанная при подключении к БД;

{ApplicationProperty:dbURL} – URL для соединения, указанная при подключении к БД;

{ApplicationProperty:dbPassword} – пароль, указанный при подключении к БД (закриптован, при необходимости используйте {DECR:{{ApplicationProperty:dbPassword}}});

{ApplicationProperty:AsyncTask} – код текущей выполняемой в асинхронном режиме задачи из списка асинхронных задач (см.п.**Ошибка! Источник ссылки не найден.**). При выполнении задачи в синхронном ежиме это свойство равно пустой строке;

{FormProperty:ProcessiongRowNumber} – номер текущей обрабатываемой строки запроса или файла;

{FormProperty:TASK_ERROR_CODE} – код ошибки, вызванной в одной из программ, закодированных в свойствах TASK_PRG_BEGIN, TASK_QUERY, TASK_PRG_ROW , TASK_PRG_END;

{FormProperty:TASK_ERROR_TEXT} – текст сообщения об ошибке, вызванной в одной из программ, закодированных в свойствах TASK_PRG_BEGIN, TASK_QUERY, TASK_PRG_ROW , TASK_PRG_END;

{FormProperty:TaskMode} – режим выполнения задачи (Async, Sync)

{FormProperty:ProcessingElapsedTime} – время истекшее с начала выполнения задачи в формате hh:mm:ss

{FormProperty:TotalRowNumber} – общее количество строк в запросе или в файле

{FormProperty:FileRecord}- содержимое текущей обрабатываемой строки считанной из файла. Если задача обрабатывает не файл, а запрос, то это свойство равно пустой строке.

Верификатор

Верификатор – это набор из двух программ, предназначенных для проверки и исправления данных, вводимых пользователем в полях форм типа BLANK. Одна из программ выполняется браузером пользователя и разрабатывается на языке JavaScript, другая – выполняет проверку введённых данных и использует метаязык LUI. Верификатор не является экранной формой. Присвоение паре программ статуса формы обусловлено тем, что для форм поддерживается версионность и генерация скриптов для обеспечения поддержки прикладных систем. Как и всякая форма, верификаторы могут иметь различные конфигурации. Однако в процессе функционирования прикладной системы используется только одна конфигурация. Имя этой конфигурации устанавливается в параметре прикладной системы VERIFIER_CONFIGURATION. По умолчанию его значение DEFAULT. LUI поставляется с заранее созданными верификаторами для всех основных типов данных. Разработанная в LUI прикладная система может не иметь собственных верификаторов – они будут унаследованы из LUI.

Пример:

Пусть прикладная система TEST в свойстве VERIFIER полей бланков содержит ссылку на верификатор LUI_V_NUMBER. При входе пользователя в прикладную систему TEST верификатор LUI_V_NUMBER будет разыскиваться среди форм прикладной системы TEST. И если он не будет найден, будет выполнен поиск в метаданных прикладной системы LUI. Следует иметь в виду, что чтение описаний верификаторов из БД выполняется при попытке входа пользователя в прикладную систему. В процессе работы пользователя с прикладной системой обновление описаний верификаторов не выполняется.

★ Общие свойства формы верификатора

Code – *Уникальный в рамках прикладной системы код формы*. Этот код вводится при создании верификатора и не меняется в разных её конфигурациях. По этому коду в совокупности с кодом конфигурации выполняется идентификация и загрузка верификатора из базы данных для выполнения в прикладной системе. Этот код указывается в свойствах VERIFIER полей бланков. Уникальный код может быть изменён в редакторе форм LUI. При этом, автоматических изменений кода в свойстве VERIFIER полей бланков не выполняется. Это надо делать вручную. Рекомендуется, чтобы код состоял из:

- Краткого обозначения прикладной системы
- Суффикса “V”
- Мнемоники отражающей смысл верификатора

Пример: LUI_V_NUMBER – верификатор для полей типа NUMBER

VERIFIER_PRG – *Программа для браузера*. В этом свойстве следует написать программу на языке JavaScript. При создании верификатора в этом свойстве генерируется прототип программы. Пример:

```
fieldVerify.TST_V_TEST = function (param) {
   // param - object with input parameters
   // Predefined parameters:
   // param.value       - field value
   // param.valuePrev   - previous field value
   // param.cursor      - cursor position
   // param.cursorPrev  - previous cursor position
  // param.userInput   - Boolean user input indicator: true - value entered by the
  user; false - value sent from the server
   var ret = // returned object
   {   exitCode:0, // returned code: =0 - success
                   // !=0 - error
       exitMessage:"OK", // additional message
```

```
            value:param.value, // output value
            cursor:param.cursor // new cursor position
        };
        return ret;
    };
```

Из этого примера видно, что это функция в объекте fieldVerify. Объект fieldVerify создаётся в браузере пользователя и в нем содержатся функции всех верификаторов нужных прикладной системе.

Функции верификатора передаётся объект – носитель входных параметров. Предопределёнными параметрами являются:

- param.value - текущее содержимое поля;
- param.valuePrev - предыдущее содержимое поля, установленное при последнем успешном выполнении функции;
- param.cursor - текущая позиция курсора (измеряется начиная с нуля);
- param.cursorPrev - предыдущая позиция курсора, установленная при последнем успешном выполнении функции;
- param.userInput - логический параметр – true – содержимое поля изменено пользователем, false – содержимое поля изменено сервером приложений

Функция верификатора обязана вернуть объект, в котором передаются выходные параметры:

- exitCode - код возврата (будет использоваться в следующих версиях, сейчас всегда должен быть равен 0);
- exitMessage - текст сообщения (будет использоваться в следующих версиях, сейчас всегда должен быть равен “OK”);
- value - новое содержимое поля
- cursor - новое положение курсора

VERIFIER_SRV_PRG – *Программа для LUI*. Здесь можно ввести программу - выражение (надо заключать в фигурные скобки) для вычисления кода ошибки, выполняемую сервером LUI. Эта программа может использовать любые языки доступные в LUI. Эта программа выполняется после верификатора выполняемого в браузере, после применения т.н. "алфавита" и после конвертирования значения поля к целевому типу данных по заданному формату. Выражение должно вычислить код сообщения, которое существует в справочнике сообщений или пустую строку или NULL. Пустая строка или NULL будут восприняты как подтверждение правильности данных введённых в поле.

Примеры:

```
{SQL:(LUI$) select 'LUI-99999' where {F2SQL:{Property:Code}}<to_date('1900','YYYY');}
```
- проверка того, что введена дата превосходящая или равная 1 января 1900 года

```
{JS:if ({Property:Value}<0) "LUI-88888";}
```
- проверка того, что введено положительное число или ноль

★ Дополнительные параметры верификатора

JavaScript – функция верификатора, выполняемая в браузере пользователя, может нуждаться в дополнительных параметрах. Например, формат данных поля. Для передачи их через объект “param” необходимо создать входные параметры верификатора. Эти параметры имеют свойства аналогичные входным параметрам форм списков и бланков. Однако, следует иметь ввиду, что никакая вызывающая форма не передаст их через свои выходные параметры действия. Значение параметра должно быть вычислено в самом верификаторе в свойстве VALUE входного параметра.

Пример:

```
{JS: if ("{Property:FORMAT}")"{Property:FORMAT}";else "dd.mm.yyyy hh24:mi:ss";}
```

Если код входного параметра равен “FORMAT”, то в JavaScript функции к этому параметру следует обращаться как “param.FORMAT”.

Шаблонные формы

Шаблонные формы не предназначены для отображения в интерфейсе пользователя. Они являются носителями значений по умолчанию для свойств форм и элементов форм. Кроме того шаблонные формы могут содержать шаблонные действия автоматически добавляемые в каждую прикладную форму при её загрузке для выполнения в сеансе пользователя прикладной системы.

Прикладная система может обладать следующими шаблонными формами:

- LIST_TEMPLATE – шаблон для всех форм типа “LIST”
- BLANK_TEMPLATE – шаблон для всех форм типа “BLANK”
- TREE_TEMPLATE - шаблон для всех форм типа “LIST” отображаемых в виде иерархии
- MENU_TEMPLATE - шаблон для всех форм типа “MENU”

Если в прикладной системе отсутствует какая-либо из этих форм, то в качестве шаблона будет применяться соответствующая форма из прикладной системы LUI.

В прикладной системе LUI эти формы расположены в группе LUI_TEMPLATES. Если разработчик приложения решил создать собственные шаблонные формы в своём приложении, то рекомендуется создать группу с подобным названием их хранить шаблоны в ней.

Шаблонные формы считываются в момент попытки входа пользователя в приложение перед выдачей на экран первой формы. В процессе работы пользователя в прикладной системе обновление данных о шаблонных формах для уже начавшегося сеанса не выполняется.

В каждой прикладной системе имеется параметр TEMPLATE_CONFIGURATION, который указывает серверу приложений код считываемой конфигурации для всех шаблонных форм.

LIST_TEMPLATE

Эта форма является шаблоном для списков. Она предназначена для :

- генерации системного меню уровня формы для всех списков приложения
- генерации системного меню уровня столбца для всех списков приложения
- генерации действий активных для ячеек списка в режиме QBE
- установки значений по умолчанию для свойств столбцов

Рассмотрим все возможные типы элементов этой шаблонной формы и правила их кодирования.

★ Пункты системного меню уровня списка

Действия, которые добавляются к прикладному списку в качестве пунктов системного меню уровня формы, должны располагаться на уровне формы, то есть не должны быть включены в какую-либо группу и не должны быть подчинены какому-либо столбцу. Их код должен начинаться с префикса “SYSMENU$”.

Вызов системного меню при работе прикладного списка зависит от реализации сервера визуализации и значения параметра APPEARANCE прикладной системы. Как правило, системное меню вызывается кликом левой кнопки мыши на иконку в левом верхнем углу окна формы. Иногда требуется сделать форму, у которой нет системного меню. Для этого используется команда метаязыка LUI:

FormProperty:Var SysMenu=N

Эту команду следует применять в свойстве формы ON_LOAD .

Если разработчик формы намерен скрыть или переопределить поведение пункта системного меню, то в прикладной форме необходимо создать действе с кодом, совпадающим с кодом пункта системного меню и для созданного действия установить нужные значения свойств.

★ Пункты системного меню уровня столбца

Действия, которые добавляются к прикладному списку в качестве пунктов системного меню уровня столбца, должны располагаться на уровне формы, то есть не должны быть включены в какую-либо группу и не должны быть подчинены какому-либо столбцу. Их код должен начинаться с префикса “COLMENU$”.

В момент загрузи прикладной формы для её выполнения, к этой форме добавляются действия, закодированные в шаблонном списке как пункты системного меню столбца. Привязка этих действий к каждому столбцу не выполняется при загрузке формы. Вместо этого для каждого пункта устанавливается динамическое значение свойства Parent.

Property:Parent= {**FormProperty**:Var ColumnMenu}

Поэтому принадлежность этих действий определяется динамически в момент инициализации системного меню уровня столбца.

Вызов системного меню уровня столбца при работе прикладного списка зависит от реализации сервера визуализации и значения параметра APPEARANCE прикладной системы. Как правило, системное меню уровня столбца вызывается кликом левой кнопки мыши на имя столбца. Иногда требуется сделать форму, у которой нет системного меню для какого-либо столбца. В этом случае, необходимо в прикладном списке создать копии всех действий с префиксом “COLMENU$” из шаблонного списка и установить нужные значения их свойств. Например, в свойстве ACTIVE запрет отображения элемента меню для столбца с кодом COL1 можно реализовать так:

```
{JS:if ("{FormProperty:Var ColumnMenu}"=="COL1")"N";
 else { var sort=0{ElementProperty:{FormProperty:Var ColumnMenu}.SORT};
        if (sort!=1) "Y";  else "N";
       }
}
```

★ Действия в ячейках списка доступные в режиме QBE

Эти действия также располагаются на уровне формы. Их код должен начинаться с префикса “QBE$ITEM”.

Шаблонная форма имеет три таких действия:

QBE$ITEM_LOV – для вызова списка типовых значений столбца в режиме QBE по клавише F9

QBE$ITEM – для вызова бланка облегчённого ввода критерия поиска по клавише Enter

QBE$ITEM_DUPLICATE – для дублирования критерия поиска в текущую ячейку и ячейки предыдущей строки по клавише F4

★ Значения свойств столбцов по умолчанию

Для хранения значений свойств по умолчанию в шаблонной форме должны быть созданы столбцы для каждого типа данных LUI. Тип данных шаблонного столбца должен соответствовать типу данных столбца, для которого он является шаблоном. Допускается наличие для одного типа данных более одного шаблона столбца списков. В каждом из шаблонов будет свой набор свойств. Например, формат, стиль отображения, выражение для подсветки и т.п. В этом случае при создании столбца прикладного списка разработчик будет иметь возможность выбора шаблона.

Коды столбцов рекомендуется задавать следующим образом: ITEM_<код типа данных>. Например, ITEM_NUMBER – шаблон столбца типа NUMBER. Если требуется более одного шаблона, следует добавить постфикс нагруженный смыслом. Пример: ITEM_NUMBER_INT – шаблон для целых чисел.

BLANK_TEMPLATE

Эта форма является шаблоном для списков. Она предназначена для:

- генерации системного меню для всех бланков приложения
- установки значений по умолчанию для свойств полей бланка
- генерации действий уровня поля бланка для вызова списков допустимых значений
- генерации действий уровня поля бланка для вызова форм облегчённого/расширенного редактирования данных

Рассмотрим все возможные типы элементов этой шаблонной формы и правила их кодирования.

★ Пункты системного меню бланка

Логика и способ кодирования совпадают с ***Пункты системного меню уровня списка***

★ Значения свойств полей по умолчанию

Для хранения значений свойств по умолчанию в шаблонной форме должны быть созданы поля для каждого типа данных LUI. Тип данных шаблонного поля должен соответствовать типу данных поля, для которого он является шаблоном. Допускается наличие для одного типа данных более одного шаблона поля бланков. В каждом из шаблонов будет свой набор свойств. Например, формат, стиль отображения, выражение для подсветки и т.п. В этом случае при создании поля прикладного бланка разработчик будет иметь возможность выбора шаблона.

Коды полей рекомендуется задавать следующим образом: FIELD_<код типа данных>. Например FIELD_NUMBER – шаблон поля типа NUMBER. Если требуется более одного шаблона следует добавить постфикс нагруженный смыслом. Пример: FIELD_NUMBER_INT – шаблон для целых чисел.

★ Вызов списков допустимых значений в полях бланков

За вызов списков допустимых значений в полях бланков отвечает действие с кодом LOV$FIELD. Действие размещено на уровне формы.

В момент загрузки прикладного бланка, для каждого поля у которого есть заполненное свойство VARIANTS, но нет собственного действия с ролью F9 (LOV), выполняется создание дополнительного действия с кодом равным LOV$<код поля>.

★ Вызов форм облегчённого или расширенного редактирования данных в полях бланка

За вызов форм редактирования данных в полях бланка отвечают действия шаблонного бланка, размещённое в соответствующем шаблонном поле. Коды этих действий должны быть заданы по следующему правилу: EDITOR$<код шаблонного поля>. Например, для шаблонного поля FIELD_DATETIME создано действие EDITOR$FILED_DATETIME, которое вызывает календарь.

В момент загрузки прикладного бланка, для каждого поля у которого нет собственного действия с кодом, начинающимся на “EDITOR$”, выполняется создание дополнительного действия с кодом равным EDITOR$<код поля>.

TREE_TEMPLATE

Эта шаблонная форма является носителем действий, которые образуют системное меню списка отображаемого в виде иерархии.

Логика и способ кодирования действий совпадают с ***Пункты системного меню уровня списка***

MENU_TEMPLATE

Эта шаблонная форма является носителем действий, которые образуют системное меню форм типа “MENU”.

Логика и способ кодирования действий совпадают с ***Пункты системного меню уровня списка***

Формы группы LUI_COMMON

В прикладной системе LUI эти формы расположены в группе LUI_COMMON. Это формы, которые могут вызваться из других приложений так, как если бы они были созданы внутри самих приложений. Сервер приложений, когда возникает необходимость загрузки формы из метаданных, сначала разыскивает форму среди форм прикладной системы, если она не будет найдена, поиск будет продолжен среди форм группы LUI_COMMON прикладной системы LUI.

Формы группы LUI_COMMON для выполнения своих операций, как правило, не используют соединения с БД, установленные прикладной системой. Вместо этого, для своих целей эти формы используют служебное соединение с БД, в которой установлена серверная часть LUI.

Формы группы LUI_COMMON предназначены для снабжения прикладных систем инфраструктурными компонентами:

- Средства администрирования, управления пользователями и сеансами
- Дополнения форм (ассоциируемые файлы)
- Алфавиты проверки данных
- Национальные языки
- Справочник сообщений об ошибках
- Синхронное и асинхронное выполнение задач
- Параметры прикладной системы
- Формы для реализации пунктов системного меню
- Формы облегчённого и расширенного ввода данных в поля бланков
- Формы облегчённого ввода критерия поиска данных в списках режиме QBE
- Верификаторы для проверки и коррекции данных вводимых в поля бланков
- Команды ОС для:
    - выполнения SQL скриптов утилитой psql
    - удаления файлов и каталогов
    - выполнения резервного копирования схемы данных LUI

Типы данных

В LUI и во всех прикладных системах, создаваемых в LUI все поля бланков и столбцы списков являются строками неопределённой длины. Понятию “Тип данных” в LUI соответствует набор настроек, который определяет поведение сервера приложений по отношению к данным в БД и к данным в формах.

LUI поставляется с настроенными типами данных, которых достаточно для разработки приложений большинства предметных областей. Настройки типов данных выполняются в среде разработки приложений в пункте главного меню “Настройки->Типы данных”. Настройки типов данных привязаны к прикладной системе и наследуются из прикладной системы LUI. Пример: если для прикладной системы TEST нет описания типа данных NUMBER, то его настройки наследуются из прикладной системы LUI. Если для прикладной системы TEST есть описание типа данных NUMBER, то все настройки пусть даже не полные будут браться из TEST. Следует иметь в виду, что чтение описаний типов данных из БД выполняется при попытке входа пользователя в прикладную систему. В процессе работы пользователя с прикладной системой обновление описаний типов данных не выполняется.

Понятие “Тип данных” существует в СУБД. Для определённости типы данных в СУБД будем называть “тип данных СУБД” а тип данных, настроенный в прикладной системе – “тип данных LUI”.

★ Типы данных СУБД

Ввод и изменение типов данных имеющихся в СУБД выполняется в пункте “Типы данных-Типы данных СУБД” главного меню.

★ Общие свойства типа данных LUI

Для каждого типа данных LUI необходимо указать следующие основные свойства:

- Код прикладной системы
- Уникальный код типа данных
- Название типа данных (желательно на всех языках прикладной системы)
- Признак применимости форматов данных и алфавитов. Форматы данных применимы, например, к числам и датам, но не применяются для символьных строк. Возможные значения:
  - F – применимы форматы
  - A – применимы алфавиты
  - N – формат и алфавит не применим

Для типа данных LUI следует указать необходимость преобразований к строке в разных фразах оператора SELECT языка SQL.

- Во фразе **SELECT** – управляет построением запроса к данным в списках и бланках. Для не текстовых данных, как правило, должно быть Y (Да)
- Во фразе **WHERE** - управляет необходимостью преобразования к строке типа данных СУБД при построении предикатов поиска. В большинстве случае должно быть N(нет) для сохранения поиска по индексам
- Во фразе **ORDER BY** – управляет необходимостью преобразования к строке типа данных СУБД при построении фразы “order by”. В большинстве случае должно быть N(нет) для сохранения индексной сортировки

★ Настройка обработки пустых строк и NULL-значений

Для типа данных LUI следует указать, как сервер приложений должен относиться к пустой строке и отсутствию значения при извлечении данных из СУБД и при преобразовании незаполненного поля бланка к данным передаваемым в СУБД. Это особенно важно для текстовых типов данных. Так, например в Oracle пустая строка эквивалентна NULL и запрос `select 'true' from dual where '' is null` вернёт ‘true’. В PostgreSQL запрос `select 'true' where '' is null` не даст данных.

Возможны следующие режимы:

- **Пустое поле в LUI -> NULL в СУБД, NULL в СУБД->Пустое поле в LUI.**

В этом режиме пустая строка значения поля в LUI будет преобразована к NULL при передаче данных в СУБД. NULL извлекаемый из БД будет отображаться в пустое, незаполненное поле бланка и пустую ячейку списка

- **Пустое поле в LUI-> пустая строка в СУБД, Пустая строка в СУБД->Пустое поле в LUI.**
  В этом режиме пустая строка значения поля в LUI будет преобразована к пустой строке (‘’) при передаче данных в СУБД. Пустая строка (‘’) извлекаемая из БД будет отображаться в пустое, незаполненное поле бланка и пустую ячейку списка. Этот режим будет полезен для приложений, в которых все текстовые столбцы таблиц имеют ограничение NOT NULL.

Если выбран первый режим, то для отображения пустых строк в полях бланков и ячейках списков необходимо ввести

- **Специальное обозначение пустой строки**

  Это один или несколько символов, которые обозначают пустую строку для данного типа данных LUI в прикладной системе. Вводя это обозначение, следует использовать символы, появление которых наименее вероятно в прикладных данных исходя из предметной области прикладной системы. Для не текстовых типов данных вводить специальное обозначение пустой строки не обязательно.

Если выбран второй режим, то для отображения NULL-значений в полях бланков и ячейках списков необходимо ввести

- **Специальное обозначение для NULL**
  Это один или несколько символов, которые обозначают NULL-значение в прикладной системе. Вводя это обозначение, следует использовать символы, появление которых наименее вероятно в прикладных данных исходя из предметной области прикладной системы.

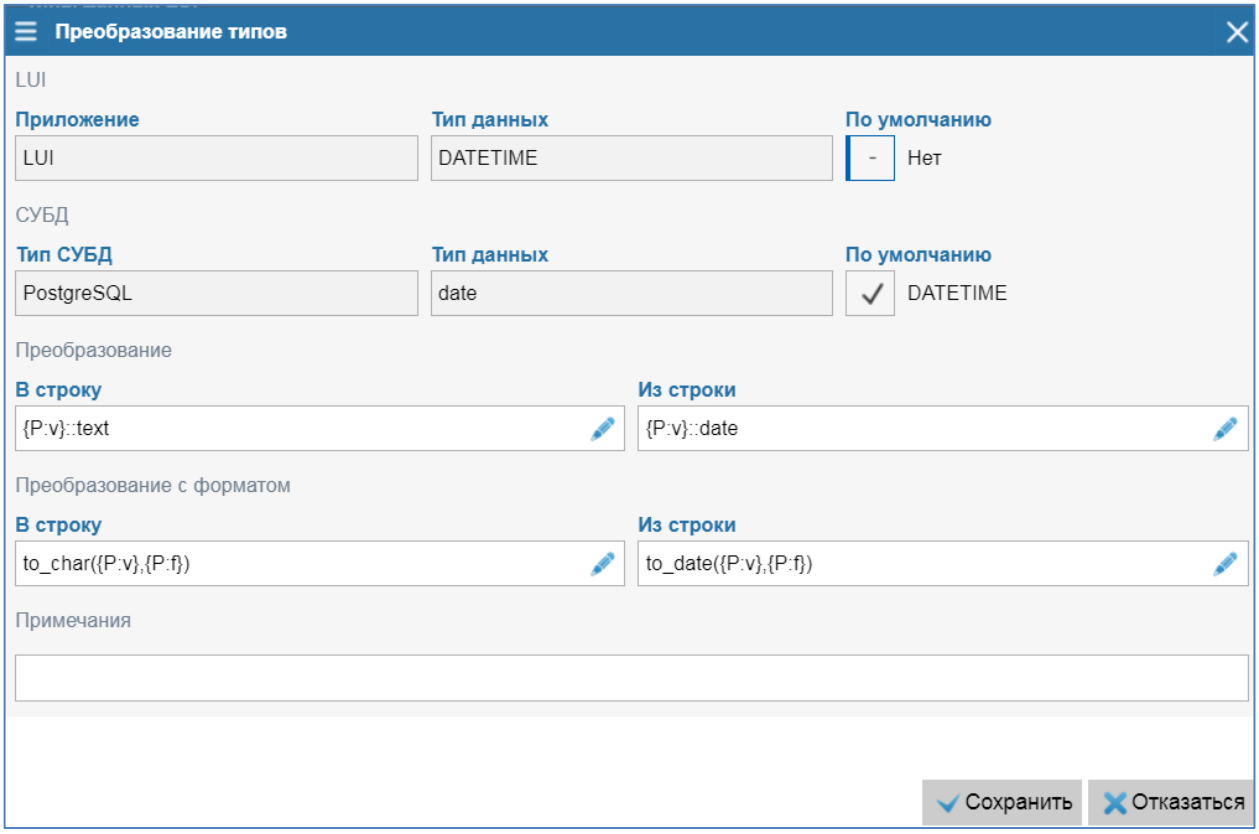
★ Настройка конвертора данных LUI<->СУБД

Для каждого типа данных LUI необходимо настроить правила преобразования данных, извлекаемых из СУБД, в строку (в LUI все данные это строки) и правила преобразования строки к данным СУБД. При этом если тип данных LUI должен поддерживать формат, то необходимо ввести две пары преобразований – для случая без форматного использования и для случая с форматированием. Правила преобразований существенно зависят от СУБД. Нескольким типам данных СУБД может соответствовать один или несколько типов данных LUI. Обратное утверждение тоже верно: нескольким типам данных LUI может соответствовать один или несколько типов данных СУБД. При вводе преобразований СУБД<->LUI следует стремиться ввести их как можно больше и полнее. Несмотря на то, что типы данных СУБД и LUI относятся друг к другу как многие ко многим для каждого типа данных LUI следует указать в какой тип данных СУБД он будет преобразовываться по умолчанию. Это необходимо для обеспечения работы оператора F2SQL (см.***F2SQL - значение элемента формы сконвертированное для языка SQL***). Точно также для типа данных СУБД необходимо указать, в какой тип данных LUI он отображается по умолчанию. Это необходимо для автоматического рекомендательного подбора типа данных LUI для полей бланков и столбцов списков при создании форм, основанных на запросах к данным (select … from … where…).

Настройка преобразований типов данных СУБД в строку и из строки (т.е. в LUI и из LUI) выполняется с помощью формул, которые представляют собой выражения на метаязыке LUI. Для упрощения нотации можно использовать следующие динамические свойства элементов форм, которые формируются сервером приложений в момент необходимости конвертации данных.

- {**P**:v} – Для преобразования LUI->СУБД это содержимое поля бланка или ячейки списка заключённое в апострофы с заменой апострофов внутри текстовой строки парой апострофов. Т.е. если поле бланка содержит текст “O’Reilly” то вместо {P:v} будет ‘O’’Reilly’. Для преобразования СУБД->LUI это псевдоним столбца запроса.
- {**P**:f} – Формат данных поля бланка или столбца списка. При этом форматная строка заключена в апострофы с заменой апострофов внутри текстовой строки формата парой апострофов.

Пример преобразования типа данных LUI “DATETIME” и типа данных СУБД “date”.



В этой форме указано следующее:

- Для прикладной системы LUI тип данных LUI “DATETIME” по умолчанию не отображается в тип данных “date” СУБД PostgreSQL (в самом деле, для этого есть более подходящий – timestamp).
- Для СУБД PostgreSQL тип данных “date” по умолчанию отображается в тип данных LUI “DATETIME”. При этом если формат данных поля бланка или столбца списка не указан, то это преобразование выполняется выражением {**P**:v}::text, а если формат указан, то – выражением to_char({**P**:v},{**P**:f}). При преобразовании из LUI в СУБД в случае отсутствия формата будет применяться выражение {**P**:v}::date, а при наличии формата – выражение to_date({**P**:v},{**P**:f}).

Для более наглядного представления введённых преобразований LUI<->СУБД можно использовать пункт главного меню “Типы данных-Матрица Типов”. В списке, который отображает матрицу соответствия типов данных , в ячейках возможны следующие значения:

- -- введены преобразования, но тип данных СУБД не отображается в тип данных LUI по умолчанию, и тип данных LUI не отображается в тип данных СУБД по умолчанию
- -D введены преобразования, но тип данных СУБД не отображается в тип данных LUI по умолчанию, а тип данных LUI отображается в тип данных СУБД по умолчанию
- D- введены преобразования, и тип данных СУБД отображается в тип данных LUI по умолчанию, а тип данных LUI не отображается в тип данных СУБД по умолчанию
- DD -- введены преобразования, и тип данных СУБД отображается в тип данных LUI по умолчанию, и тип данных LUI отображается в тип данных СУБД по умолчанию

★ Настройка языка QBE

QBE – это упрощённый язык задания запросов к БД с использованием образцов значений полей в виде текстовой строки. QBE – аббревиатура “**Q**uery **B**y **E**xample”. Реализации QBE в списках LUI преобразуют пользовательский ввод в формальный запрос к базе данных, что позволяет конечному пользователю создавать сложные запросы без необходимости изучать SQL. В Списке остаются только те строки, которые соответствуют текстовым образцам, заданным в полях, на которые необходимо наложить ограничения. Условия, заданные таким способом, легко читаются и понимаются пользователем, так как интуитивно понятны и наглядны. Кроме того, установленные условия легко подвергаются коррекции, модификации, сохранению под именем и быстрому восстановлению в нужный момент.

В каждом списке, созданном в LUI, по умолчанию есть возможность перехода в режим запроса.

Это делается либо иконкой  в панели инструментов, либо пунктом “Ввод запроса” системного меню списка, либо нажатием F7. В Конкретных реализациях сервера отображений способ перехода в режим запроса может отличаться от описанного выше.

В этом режиме список очищается от данных, и пользовать может вводить тексты в любое видимое поле одной или нескольких строк.

Текст в любой ячейке преобразуется в одно или несколько условий (предикатов), применяемых к тому столбцу, в котором введён текст. Условия, указанные в одной строке объединяются отношением "И". Несколько строк с условиями объединяются отношением "Или".

Выполнение запроса инициируется либо иконкой  в панели инструментов, либо пунктом “Выполнение запроса” системного меню списка, либо нажатием F8.

Пример:

№ договора	Дата	Клиент	Юр. ст.	Конвертов	Объектов
%000%	>=01.01.11				
	<01.01.11		ЮЛ		
					3

Данное условие интерпретируется так:

Отобразить все договоры, которые в номере содержат три нуля подряд и заключены в 2011 году, а также договоры с юридическими лицами, заключёнными до 2011 года и кроме того, любые договоры на ровно три объекта.

На уровне SQL условия из данного примера порождают следующие предикаты:

```
CONTRACT_NO like '%000%' and CONTRACT_DATE>=To_DATE('01.01.11','DD.MM.YY')
or CONTRACT_DATE<To_DATE('01.01.11','DD.MM.YY') and LEGAL_STATUS='ЮЛ'
or N_OBJECT=3
```

Обратите внимание на специальные символы, введённые в этом запросе - “%”, “>=”, ”<”. Это не что иное, как обозначение операторов сравнения языка QBE. Операндами языка QBE являются столбцы запроса и текстовые константы, введённые пользователем, с которыми сравниваются их значения на уровне СУБД. Помимо символов обозначающих операторы сравнения, в реализации языка QBE в LUI есть оператор отрицания. Оператор отрицания кодируется символом “!” (восклицательный знак) и настройке не подлежит. В случае необходимости инверсии оператора сравнения QBE, оператор отрицания должен вводиться непосредственно перед оператором сравнения. Например: “!=” эквивалентно НЕ РАВНО. Реализации языка QBE в LUI предполагает, что могут существовать операторы сравнения следующих видов:

- Оператор с одним операндом. Пример: “столбец” is null
- Оператор с двумя операндами. Пример: “столбец” = 123.01
- Оператор с тремя операндами. Пример: “столбец” between 123.01 and 123.09

В процессе построения оператора SELECT языка SQL операторы языка QBE преобразовываются в функции и операторы сравнения языка SQL, а текстовые строки введённые пользователем преобразовываются к типам данных соответствующих столбцов запроса. Эти преобразования и состав операторов сравнения существенно зависят от типа СУБД и типа данных СУБД.

Помимо возможности ввода критериев непосредственно в списке, LUI предоставляет возможность облегчённого ввода критерия в специальном бланке. Обычно он вызывается нажатием Enter в требуемом столбце (или правая кнопка мыши и выбор пункта “Ввод критерия поиска”), когда список находится в режиме запроса.

Для каждого типа данных LUI необходимо настроить набор операторов сравнения языка QBE, для обеспечения пользователя возможностью ввода необходимых критериев поиска.

LUI поставляется с уже настроенными операторами, но архитектор того или иного приложения может сделать собственные операторы на основе имеющихся в LUI.

Свойства оператора QBE, влияющие на интерфейс пользователя

Для оператора языка QBE необходимо ввести следующие свойства, влияющие на интерфейс конечного пользователя:

- **№п/п**

  Порядковый номер для отображения списка операторов в интерфейсе пользователя

- **Название**
  Название оператора на всех языках приложения
- **Обозначение в QBE**
  Обозначение оператора в языке QBE. Это должна быть строка из одного, двух, желательно не более трёх символов, которые пользователь будет вводить в списках в режиме QBE для задания оператора сравнения. Требуется уникальность обозначения в QBE в рамках типа данных LUI
- **Иконка**
  Иконка используется в бланке облегчённого ввода QBE. Бланк вызывается нажатием Enter в режиме QBE. Требуется уникальность иконки в рамках типа данных LUI

Свойства оператора QBE, влияющие на обработку данных

Для оператора языка QBE необходимо ввести следующие свойства, влияющие на обработку данных вводимых в списках в режиме QBE и данных вводимых в бланке облегчённого ввода QBE:

- **Код в SQL**

  Это код оператора сравнения в языке SQL

- **Тип СУБД**

  Если оператор применим во всех СУБД поддерживаемых LUI, то следует ввести “Любая”

  Если оператор применим только в конкретной СУБД, то следует явно её указать.

  Если оператор применим в двух СУБД, но не применим в остальных, следует создать ещё одно описание оператора QBE

- **Ранг**

  Это порядковый номер оператора QBE в последовательности выявления оператора исходя из текста введённого пользователем.

- **Выражение идентификации**

Это выражение на метаязыке LUI, которое должно вернуть символ ‘Y’ если текст, введённый пользователем в ячейку списка в режиме QBE, соответствует данному оператору языка QBE, и символ ‘N’ – в противном случае. В выражении можно ссылаться на переменную {**P**:v}, которую сервер приложений автоматически создаёт при необходимости интерпретации QBE. Эта переменная содержит текст критерия, введённый пользователем лишённый символа оператора отрицания. Тот факт, что переменная лишена оператора отрицания, позволяет упростить написание выражения идентификации. Помимо этой переменной, в выражении идентификации можно использовать переменную {**P**:QBEsign}. Она содержит обозначение оператора QBE. Пример выражения идентификации на JavaScript:

{**JS**:if("{**P**:v}".search(/^{**P**:QBEsign}.+/)==0)"Y"; else "N";}

Это интерпретируется так: если текст введённый пользователем начинается с символов указанных в свойстве “**Обозначение в QBE”** и за этими символами следует хотя бы один любой символ, то выражение идентификации равно “Y”.

- **Конструктор SQL-условия**

Это выражение на метаязыке LUI, которое должно вернуть текст предиката оператора SELECT языка SQL. Этот конструктор используется сервером приложений для построения запросов к БД в списках. В выражении можно ссылаться на следующие переменные:

- **{P:QBEcolumn}** - псевдоним столбца
- **{P:QBEcolumn2str}** - псевдоним столбца обёрнутый функцией преобразования к строке
- **{P:QBEnot}** - отрицание (если оно есть, то содержит "not "(пробел в конце) иначе пусто)

- **{P:QBEnotSign}** - отрицание (если оно есть, то содержит "!" иначе пусто)
- **{P:QBEcriteria}** - критерий поиска, обёрнутый преобразованием к типу данных столбца
- **{P:QBEcriteria2}** - 2-я часть критерия поиска, обёрнутая преобразованием к типу данных столбца это свойство заполняется, только если в конструкторе предиката есть ссылка на него. 2-я часть вычисляется исходя из того, что знак оператора разделяет первую и вторую части Пример: 10><20 для between 10 and 20, где "><" – оператор "между" языка QBE.
- **{P:QBEsign}** - обозначение оператора сравнения языка QBE
- **{P:QBEoperator}** - оператор сравнения языка SQL
- **{P:QBEcriteriaRaw}** - исходный критерий поиска, введённый пользователем, лишённый отрицания и знака оператора
- **{P:QBEcriteriaRaw2}** - исходная 2-я часть критерия поиска, введённая пользователем
- **{P:QBEnss}** – (null special symbol) символ для отображения и ввода null-значений. Пустая строка если null-значения отображаются и вводятся в интерфейсе как пустая строка
- **{P:QBEess}** – (empty string special symbol) - символ для отображения и ввода пустой строки. Пустая строка если пустая строка отображается и вводится в интерфейсе как пустая строка.

- **Конструктор QBE-условия**

Это выражение на метаязыке LUI, которое должно вернуть текст условия поиска на языке QBE. Этот конструктор используется сервером приложений для построения QBE-условия при работе пользователя в бланке упрощённого ввода условий. В выражении можно ссылаться на следующие переменные:

- **{P:QBEnot}** - отрицание (если оно есть, то содержит "not "(пробел в конце) иначе пусто)
- **{P:QBEnotSign}**- отрицание (если оно есть, то содержит "!" иначе пусто)
- **{P:QBEcriteria}** - критерий поиска, введённый в поле
- **{P:QBEcriteria2}** - критерий поиска, введённый во втором поле
- **{P:QBEsign}** - знак оператора сравнения, введённый пользователем
- **{P:QBEoperator}** - оператор сравнения языка SQL
- **{P:QBEnss}** - null special symbol- символ для отображения и ввода null-значений. Пустая строка если null-значения отображаются и вводятся в интерфейсе как пустая строка
- **{P:QBEess}** - empty string special symbol - символ для отображения и ввода пустой строки. Пустая строка если пустая строка отображается и вводится в интерфейсе как пустая строка

Алгоритм построения предикатов оператора SELECT для запроса к БД из списков

При построении текста оператора SELECT языка QBE сервер приложений анализирует все ячейки списка, в которых пользователь ввёл критерии поиска. Условия, указанные в одной строке объединяются отношением "И". Несколько строк с условиями объединяются отношением "Или". При построении условия для каждой заполненной ячейки списка, сервер приложений просматривает список операторов сравнения языка QBE, настроенных для типа данных указанного для столбца, к которому относится ячейка. Просмотр выполняется в порядке возрастания значений, указанных в поле "**Ранг**". На каждой очередной итерации просмотра выполняется вычисление выражения из поля "**Выражение идентификации**". Если выражение возвращает Y, то считается, что нужный оператор сравнения языка QBE определён. Если в результате полного просмотра всех операторов, оператор не был определён, то считается, что оператором является оператор с минимальным значением в поле "**№п/п**" (это не опечатка!). И в этом случае если обозначение этого оператора отсутствует в начале текста введённого пользователем, то этот текст дополняется слева этим обозначением. Далее считается, что этот текст введён пользователем. После определения оператора сравнения, вычисляется выражение, заданное в поле " **Конструктор SQL-условия**" этого оператора.

Перечень основных операторов сравнения языка QBE

LUI поставляется с уже настроенными операторами сравнения языка QBE. Ниже приведён их перечень.

Тип данных LUI	№п/f	Оператор QBE	Оператор SQL	Название
BOOLEAN	1	Y	Is true	истина
BOOLEAN	2	N	Is false	ложь
BOOLEAN	3		is null	пусто
CHAR	1	=	=	равно
CHAR	2	>	>	больше
CHAR	3	<	<	меньше
CHAR	4	>=	>=	больше или равно
CHAR	5	<=	<=	меньше или равно
CHAR	6	<>	<>	не равно
CHAR	7	%	like	похож на
CHAR	8	><	between	между
CHAR	9	*	similar to	подобен
CHAR	10	@@	@@	полнотекстовый поиск
CHAR	11	?	ilike	регистронезависимо похож на
CHAR	12	~	~	соответствует регулярному выражению
CHAR	13	~*	~*	регистронезависимо соответствует регулярному выражению
CHAR	14		is null	пусто
CIDR	1	=	=	равно
CIDR	2	>	>	больше
CIDR	3	<	<	меньше
CIDR	4	>=	>=	больше или равно
CIDR	5	<=	<=	меньше или равно
CIDR	6	<>	<>	не равно
CIDR	7	%	like	похож на
CIDR	8	><	between	между
CIDR	10		is null	пусто
CIDR	20	<<	<<	содержится в
CIDR	21	<<=	<<=	равен или содержится в
CIDR	22	>>	>>	содержит
CIDR	23	>>=	>>=	равен или содержит
CIDR	24	&&	&&	содержит или содержится в
DATETIME	1	=	=	равно
DATETIME	2	>	>	больше
DATETIME	3	<	<	меньше
DATETIME	4	>=	>=	больше или равно
DATETIME	5	<=	<=	меньше или равно
DATETIME	6	<>	<>	не равно
DATETIME	7	%	like	похож на
DATETIME	8	><	between	между
DATETIME	9		is null	пусто
DTRANGE	1	=	=	равно
DTRANGE	2	>	>	больше

DTRANGE	3	<	<	меньше
DTRANGE	6	<>	<>	не равно
DTRANGE	10		is null	пусто
DTRANGE	20	>>	>>	строго справа от
DTRANGE	21	<<	<<	строго слева от
DTRANGE	22	<@	<@	содержится в диапазоне
DTRANGE	23	&>	&>	не простирается левее
DTRANGE	24	&<	&<	не простирается правее
DTRANGE	25	&&	&&	пересекает
DTRANGE	26	@>	@>	содержит диапазон
INET	1	=	=	равно
INET	2	>	>	больше
INET	3	<	<	меньше
INET	4	>=	>=	больше или равно
INET	5	<=	<=	меньше или равно
INET	6	<>	<>	не равно
INET	7	%	like	похож на
INET	8	><	between	между
INET	10		is null	пусто
INET	20	<<	<<	содержится в
INET	21	<<=	<<=	равен или содержится в
INET	22	>>	>>	содержит
INET	23	>>=	>>=	равен или содержит
INET	24	&&	&&	содержит или содержится в
INTERVAL	1	=	=	равно
INTERVAL	2	>	>	больше
INTERVAL	3	<	<	меньше
INTERVAL	4	>=	>=	больше или равно
INTERVAL	5	<=	<=	меньше или равно
INTERVAL	6	<>	<>	не равно
INTERVAL	7	%	like	похож на
INTERVAL	8	><	between	между
INTERVAL	9		is null	пусто
INTERVAL_DS	1	=	=	равно
INTERVAL_DS	2	>	>	больше
INTERVAL_DS	3	<	<	меньше
INTERVAL_DS	4	>=	>=	больше или равно
INTERVAL_DS	5	<=	<=	меньше или равно
INTERVAL_DS	6	<>	<>	не равно
INTERVAL_DS	7	%	like	похож на
INTERVAL_DS	8	><	between	между
INTERVAL_DS	9		is null	пусто
MACADDR	1	=	=	равно
MACADDR	2	>	>	больше
MACADDR	3	<	<	меньше
MACADDR	4	>=	>=	больше или равно
MACADDR	5	<=	<=	меньше или равно
MACADDR	6	<>	<>	не равно

MACADDR	7	%	like	похож на
MACADDR	8	><	between	между
MACADDR	9		is null	пусто
MACADDR8	1	=	=	равно
MACADDR8	2	>	>	больше
MACADDR8	3	<	<	меньше
MACADDR8	4	>=	>=	больше или равно
MACADDR8	5	<=	<=	меньше или равно
MACADDR8	6	<>	<>	не равно
MACADDR8	7	%	like	похож на
MACADDR8	8	><	between	между
MACADDR8	10		is null	пусто
NLSTEXT	1	=	=	равно
NLSTEXT	2	>	>	больше
NLSTEXT	3	<	<	меньше
NLSTEXT	4	>=	>=	больше или равно
NLSTEXT	5	<=	<=	меньше или равно
NLSTEXT	6	<>	<>	не равно
NLSTEXT	7	%	like	похож на
NLSTEXT	8	><	between	между
NLSTEXT	9	*	similar to	подобен
NLSTEXT	10	@@	@@	полнотекстовый поиск
NLSTEXT	11	?	ilike	регистронезависимо похож на
NLSTEXT	12	~	~	соответствует регулярному выражению
NLSTEXT	13	~*	~*	регистронезависимо соответствует регулярному выражению
NLSTEXT	14		is null	пусто
NUMBER	1	=	=	равно
NUMBER	2	>	>	больше
NUMBER	3	<	<	меньше
NUMBER	4	>=	>=	больше или равно
NUMBER	5	<=	<=	меньше или равно
NUMBER	6	<>	<>	не равно
NUMBER	7	%	like	похож на
NUMBER	8	><	between	между
NUMBER	9		is null	пусто
NUMRANGE	1	=	=	равно
NUMRANGE	2	>	>	больше
NUMRANGE	3	<	<	меньше
NUMRANGE	6	<>	<>	не равно
NUMRANGE	10		is null	пусто
NUMRANGE	20	>>	>>	строго справа от
NUMRANGE	21	<<	<<	строго слева от
NUMRANGE	22	&>	&>	не простирается левее
NUMRANGE	23	&<	&<	не простирается правее
NUMRANGE	24	&&	&&	пересекает
NUMRANGE	25	@>	@>	содержит диапазон

NUMRANGE	26	<@	<@	содержится в диапазоне
RICHTEXT	1	@@	@@	полнотекстовый поиск
RICHTEXT	2	%	like	похож на
RICHTEXT	3	?	ilike	регистронезависимо похож на
RICHTEXT	4	*	similar to	подобен
RICHTEXT	5	~	~	соответствует регулярному выражению
RICHTEXT	6	~*	~*	регистронезависимо соответствует регулярному выражению
RICHTEXT	7		is null	пусто
RICHTEXT	8	=	=	равно
RICHTEXT	9	>	>	больше
RICHTEXT	10	<	<	меньше
RICHTEXT	11	>=	>=	больше или равно
RICHTEXT	12	<=	<=	меньше или равно
RICHTEXT	13	<>	<>	не равно
RICHTEXT	14	><	between	между

★ Настройка алфавитов

Наряду с верификатором алфавит это ещё один способ проверки данных вводимых пользователем в поля бланка. Этот способ более простой, но менее универсальный. Проверка выполняется на сервере приложений и если данные не соответствуют требованиям алфавита, пользователю будет выдано сообщение об ошибке. Алфавиты настраиваются в прикладной системе в пункте меню “Настройки-Алфавиты”. Если какой-либо алфавит отсутствует в прикладной системе, но прикладная система его использует (он указан в свойстве FORMAT поля бланка прикладной системы), то алфавит будет применяться из прикладной системы LUI. Разработчик может использовать алфавит не только для проверки введённых данных, но и для их автоматической коррекции.

Основные свойства алфавита:

- Код – уникальный в пределах прикладной системы код. Этот код указывается в свойстве FORMAT полей бланков, если тип данных поля поддерживает алфавит (См.*Свойства полей Бланка*)
- Название – название алфавита (если прикладная система многоязычна, то название рекомендуется вводить на всех языках приложения)
- Программа дополнительной проверки – выражение на метаязыке LUI, которое должна проанализировать значение поля, и вернуть либо само значение, либо исправленное значение, либо сгенерировать ошибку средствами LUI.
  Дополнительная проверка выполняется после проверки всех основных правил
  Пример:

```
{PLPGSQL:(LUI$)
begin
If {Property:Value}>1000 or {Property:Value}<100 then
 execute lui_r_show_message('MSG-00010');
End if;
perform set_config('LUI.RETVAL','{Property:Value}',false);
end;}
```

Здесь LUI$-код служебного соединения. См. *DBConnect – подключение к базе данных*

Для алфавита можно указать набор правил, применяемых к значению поля в заданной последовательности.

Свойства правила:

- Номер в последовательности проверки
- Регулярное выражение

- Код сообщения об ошибке из справочника сообщений
- Заменяющий текст

Если значение поля не соответствует регулярному выражению и указан код сообщения, то генерируется сообщение об ошибке.

★ Поддержка национальных языков

Для ввода и отображения данных на нескольких языках в LUI предусмотрен специальный тип данных – NLSTEXT. Поля бланков этого типа отображаются в виде группы полей. По одному полю на каждый язык. В прикладной системе существует справочник национальных языков поддерживаемых прикладной системой. Он доступен из пункта главного меню “Настройки-Языки”.

Для каждого языка необходимо ввести:

- Код языка в прикладной системе
- Название на всех языках прикладной системы
- Код языка, из которого будет браться перевод текста при отсутствии перевода на данном языке
- Код региональных настроек (будет использовано в будущем для управления параметрами сеанса в СУБД)

Язык может быть объявлен действующим по умолчанию. При входе в прикладную систему будет устанавливаться язык по умолчанию. Если язык по умолчанию установлен не один или не установлен совсем, то при входе в приложение язык по умолчанию будет выбран случайным образом (первый попавшийся).

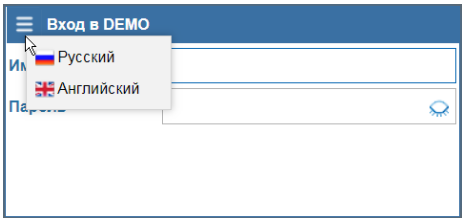
Язык может быть объявлен не действующим. В этом случае его наличие никак не сказывается на работе приложения.

Выбор текущего языка возможен только при запуске прикладной системы. (В последующих версиях LUI это может быть изменено.) Для указания языка можно использовать параметр URL Lang. Пример:

http://localhost:8080/lui2/?AppCode=LUI&Lang=ENG

Разработчик многоязычной прикладной системы должен стараться предоставить функцию выбора языка в первой отображаемой форме прикладной системы. После выбора языка прикладная система должна быть перезагружена командой ***RunApplication – запуск прикладной системы***

Обычно функция выбора языка выносится в системное меню logon-формы:



Создание форм

Перечисленные в главе «Формы » Списки, Бланки и Иерархические структуры и другие формы нужно как-то описывать, хранить эти описания и использовать для прикладных нужд. Есть два способа делать это, применяемые как по отдельности, так и последовательно друг за другом.

Первый способ – хранение описания форм (элементов и их свойств) в специальной, чётко структурированной базе данных с удобным интерфейсом для редактирования. Это способ описать вид и поведение формы в offline режиме, заранее, до начала работы прикладной системы. Такие данные, которые описывают вид и поведение формы, будем называть «***метаданные***». В runtime режиме метаданные будут загружаться в память сервера приложений и сразу начинать функционировать.

Преимущества этого способа в том, что в огромном большинстве случаев для создания работоспособных форм не нужно программировать. Достаточно описать свойства форм и их элементов.

Другой способ – создание элементов форм и описание их свойств «на лету», уже в процессе работы приложения, при помощи функций некоего процедурного языка. Преимущества этого способа в том, что можно порождать формы, набор элементов которых (да и свойства этих элементов) не могут быть определены заранее, вне функционирования приложения. Например, формы, элементы и свойства которых берутся из часто меняющегося справочника или порождаются исходя их оперативной обстановки.

Комбинированный способ состоит в том, что некая относительно статичная часть формы загружается из метаданных, а в программе, указанной в свойстве ON_LOAD формы (списка и бланка) можно воспользоваться функциями добавления оперативных элементов и/или коррекции свойств, загруженных из метаданных.

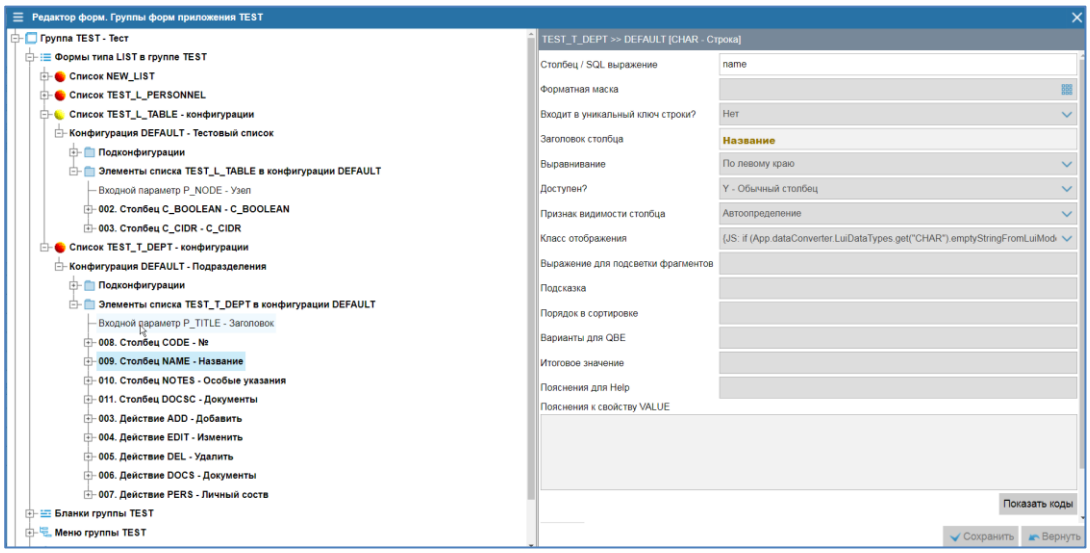
Редактор форм

Формы редактора метаданных созданы и функционируют при помощи LUI.

Для входа в редактор необходимо открыть Приложение и кликнуть на иконку «Вызов редактора форм». Отобразится древовидный навигатор редактора форм. В начале, он покажет список групп форм верхнего уровня. Раскрытие каждой группы отобразит список подгрупп и/или список форм различных типов. Каждая форма раскрывает список конфигураций. Конфигурация позволяет увидеть список подчинённых конфигураций и иерархию элементов формы.

На каждом уровне можно добавлять и изменять группы форм, формы, конфигурации форм и элементы форм. Кроме того, навигация по списку конфигураций и по элементам форм вызывает автоматическое открытие в правой части окна бланка со свойствами конфигурации или элемента.

Пример:



★ Перечень Групп форм

В узле иерархического навигатора редактора форм для каждой группы отображается слово «Группа», код группы и через тире – Наименование группы на текущем языке.

В перечне доступны следующие локальные действия (отображаемые по правой кнопке мыши):

- ***Добавить группу*** – Доступно всегда. Вызывает бланк для добавления новой группы в текущий уровень.
- ***Переименовать группу*** – Доступно только для текущего узла-группы. Вызывает бланк, где можно изменить имя группы для разных языков и переместить группу в иерархии групп.
- ***Удалить группу*** – Доступно только для текущего узла-группы. Позволяет удалить текущую (выделенную) группу.
- ***Добавить*** *{Тип формы}* – Доступно только для текущего узла-группы и только если в группе ещё нет форм этого типа. Вызывает бланк, позволяющий создать новую форму выбранного типа в текущей группе.
- ***Поиск*** – Доступно всегда. Открывает интерфейс, позволяющий искать формы по текстовым фрагментам их свойств (как по подстроке, так и по регулярному выражению).

Раскрытие узла-группы отображает подузлы по следующему принципу:

- **Списки группы** – Если в группе есть хотя бы одна форма типа Список, и нет ни бланков, ни подгрупп.
- **Бланки группы** – Если в группе есть хотя бы одна форма типа Бланк, и нет ни Списков, ни подгрупп
  …
- **Список подгрупп** – Если у группы есть хотя бы одна подгруппа, и нет ни одного Списка или бланка.
- **Список узлов-действий, состоящий из**:
  - ***Списки группы {КОД}*** – если в группе есть хотя бы один Список
  - ***Бланки группы {КОД}*** – если в группе есть хотя бы один Бланк
  …
  - ***Вложенные группы*** – если у группы есть хотя бы одна подгруппа

  Каждый их перечисленных узлов имеет локальное действие ***Открыть как список***, которое отображает содержимое узлов-действий в виде формы-Списка.

★ Перечень Форм

Может быть несколько перечней форм, сгруппированных по типам, которые отображаются при раскрытии узлов-групп или узлов-действий групп (см. «Перечень Групп форм» на стр.87). По сути – перечень Списков и Бланков определяется одной UI-формой, принимающей разные параметры. Поэтому их описание тут не разделяется.

Узлы-формы отображаются так: Слово «Список» или «Бланк» или название другого типа форм + код формы. В перечне этих узлов доступны следующие локальные действия (отображаемые по правой кнопке мыши):

- ***Добавить*** – Доступно всегда. Вызывает бланк для добавления новой формы. В процессе создания новой формы можно изменить её тип.
- ***Изменить*** – Доступно только для текущего узла-формы. Вызывает бланк для изменения атрибутов формы, включая код и подчинение группе.
- ***Удалить*** – Доступно для одного или нескольких выделенных узлов-форм. Удаляет ненужные формы.
- ***Кто вызывает*** *- N* – Доступно для текущего узла-формы. Отображает перечень (в виде иерархической структуры) всех форм, где есть действия, вызывающие текущую форму (код формы указан в свойстве COMMAND этого действия). Здесь N – количество таких форм.
- ***Аудит*** – Доступно для текущего узла-формы. Отображает Список с данными обо всех изменениях в текущей форме.

- ***Скрипт*** – Доступно для одного или нескольких выделенных узлов-форм. Генерирует текстовый набор SQL-команд для полной замены или удаления форм в метаданных.

Раскрытие узла-формы отображает подузлы-конфигурации верхнего уровня

★ Перечень Конфигураций формы

Конфигурации формы сами выстраиваются в иерархическую структуру. Раскрытие узла-формы отображает перечень конфигураций верхнего уровня, среди которых обязательно должна быть конфигурация с кодом DEFAULT. Текст узла-конфигурации состоит из слова «Конфигурация», кода конфигурации и через тире – названии конфигурации на текущем языке.

В перечне узлов-конфигураций доступны следующие локальные действия (отображаемые по правой кнопке мыши):

- ***Добавить конфигурацию*** –Доступно всегда. Вызывает бланк для добавления новой конфигурации формы.
- ***Переименовать*** {КОД} – Доступно для текущего узла-конфигурации. Вызывает бланк для изменения имени конфигурации КОД.
- ***Удалить*** {КОД} – Доступно для текущего узла-конфигурации. Удаляет конфигурацию с кодом КОД
- ***Аудит*** {КОД} – Доступно для текущего узла-конфигурации. Отображает Список с данными обо всех изменениях в конфигурации формы.
- ***Скрипт замены*** – Отображает в отдельном окне скрипт для полной замены текущей конфигурации в метаданных.
- ***Запустить*** – Запускает текущую форму в текущей конфигурации с дефолтными значениями входных параметров

Навигация по узлам-конфигурациям отображает в том же окне справа в автоматическом режиме бланк для редактирования свойств самой формы в текущей конфигурации (см. «Общие свойства Списка» на стр. 27 и «Общие свойства Бланка» на стр. 43). Подробнее о бланке для редактирования свойств форм см. «Редактирование Свойств форм» на стр.89.

Раскрытие узла-конфигурации отображает подузлы-действия:

- ***Подконфигурации*** – Отображает такой же Перечень Конфигураций, подчинённых текущей.
- ***Элементы*** {Тип} {КОД} – Отображает Перечень Элементов формы, с сохранением иерархии подчинения.

★ Перечень Элементов формы

Этот список выстраивается в естественную иерархию подчинённости, когда раскрытие узла-элемента сразу отображает перечень подузлов-элементов, непосредственно подчинённых раскрываемому элементу. Надо знать, что возможность манипулирования элементами (добавление, удаление, переименование) имеется только в иерархии, раскрываемой конфигурацией DEFAULT.

Элементы внутри любого уровня иерархии идут в такой последовательности:

- Сначала идут входные параметры формы
- Затем следуют вперемежку столбцы (для Списков), поля (для Бланков), группы и действия внутри группы типа SIDE_BY_SIDE в соответствии с Порядком (Seq)
- В самом конце идут действия – самостоятельные или из группы типа не SIDE_BY_SIDE

Для перечня действий возможны следующие манипуляции (доступные по правой кнопке мыши):

- ***Добавить элемент*** – Доступно в конфигурации DEFAULT. Отображает бланк для добавления нового элемента.
- ***Изменить*** {КОД} – Применимо к текущему элементу в конфигурации DEFAULT. Позволяет изменить Код, Название, Порядок, а также подчинённость существующего элемента формы.
- ***Удалить*** {КОД} – Доступно для одного или нескольких выделенных узлов-элементов в конфигурации DEFAULT формы. Выполняет удаление элементов формы. КОД – код текущего элемента в случае выделения одного элемента, или количество выделенных элементов, в случае выделения более одного элемента

- ***Скрипт замены*** {КОД} – Отображает в отдельном окне скрипт для полной замены элемента КОД (с подэлементами) в метаданных.
- ***Аудит*** {КОД} – Доступно для текущего узла-элемента. Отображает Список с данными обо всех действиях с элементом КОД и его свойствами.
- ***Использования*** {КОД} – Доступно для текущего узла-элемента типа Столбец, Поле или Входной Параметр. Отображает список мест (Конфигурация + Элемент + Свойство), где есть ссылка в динамическом значении на элемент КОД.

Навигация по узлам-элементам, а также множественное выделение узлов одного типа вызывает автоматическое отображение в том же окне справа бланка для редактирования в текущей конфигурации свойств одного или нескольких выделенных элементов формы. Подробнее о бланках для редактирования свойств форм и элементов форм см. «Редактирование Свойств элементов форм» на стр. 90

★ Редактирование Свойств форм

При навигации по узлам-конфигурациям в немодальном режиме возникает бланк для отображения и изменения свойств формы в текущей конфигурации. Каждое свойство отображается отдельным текстовым полем. Нажатие на кнопку «Показать коды» переключает бланк в режим отображения в качестве подписей к полям кодов свойств, а не их наименований, а кнопка «Показать названия» переключает этот режим назад.

Навигация по полям-свойствам отображает в поле «Пояснения к свойству {КОД}» текст, поясняющий суть текущего свойства.

Каждое поле-свойство является редактируемым как минимум при помощи ввода текста с клавиатуры. Кроме того, для некоторых свойств возможен выбор наиболее типичных значений при помощи выпадающего списка или LOV (кнопки справа от поля или клавиша F9).

Серый фон поля-свойства говорит о том, что своего значения этого свойства форма в данной конфигурации не имеет, а наследует его из родительской конфигурации (при наличии) или использует значение свойства по умолчанию (если редактируется форма в конфигурации DEFAULT).

Если значение поля отображается жирным бордовым цветом, то для него имеются варианты значений на разных языках. Редактирование такого поля производится посредством локального действия «Многоязычный редактор».

Для каждого поля-свойства доступны следующие локальные действия, отображаемые по клику правой кнопкой мыши в соответствующем поле:

- ***Базовое значение*** – Применимо к полям с белым фоном. Удаляет текущее значение свойства и применяет значение свойства по умолчанию. Поле при этом становится серым.
- ***Вернуть*** – Возвращает перечень всех предыдущих значений текущего свойства формы. Используются данные аудита изменений. Выбор значения из этого перечня помещает его в поле-свойство (возврат старого значения).
- ***Добавить ссылку*** – Отображает список входных параметров, столбцов (для Списков), полей (для Бланков) и некоторых системных переменных формы. Выбор из этого списка помещает ссылку на выбранный элемент в конец текущего значения свойства.
- ***Многоязычный редактор*** – Позволяет задать разные варианты значений для разных языков. В этом случае свойство станет многоязычным. Удаление значений не текущего языка делает значение моноязычным. Это действие также вызывается клавишей Enter, если значение свойство многоязычно.
- ***Скрипт*** – Отображает в отдельном окне скрипт для замены в метаданных текущего значения свойства в текущей конфигурации.
- ***В многострочное окно*** – Открывает бланк для редактирования значения свойства в многострочном поле. Для многоязычных значений бланк открывается только для просмотра. Редактирование многоязычных полей производится при помощи действия «Многоязычный редактор». Это действие также вызывается клавишей Enter, если значение свойство моноязычно.

Нажатие на кнопку «Сохранить» помещает текущие изменения в хранилище метаданных. Если текущих изменений нет, то кнопка «Сохранить» будет неактивна. Уход с текущего узла-конфигурации в иерархии редактора метаданных вызывает автосохранение сделанных изменений.

При наличии текущих изменений также доступна кнопка «Вернуть». Её нажатие отменяет все текущие изменения свойств формы.

★ Редактирование Свойств элементов форм

При навигации по узлам-элементам в немодальном режиме, а также при выделении нескольких однотипных пунктов в уровне, возникает бланк для отображения и изменения свойств элементов формы в текущей конфигурации.

Каждое свойство отображается отдельным текстовым полем. В случае выделения одного элемента слева, в этих полях отображаются значения свойств этого элемента. В случае выделения более одного элемента слева, поля свойств с различающимися значениями для выделенных элементов закрыты серой сеткой. Изменение значения свойства, даже если ранее оно различалось для выделенных элементов, приводит к установлению этого значения для всех этих элементов сразу!

Нажатие на кнопку «Показать коды» переключает бланк в режим отображения в качестве подписей к полям кодов свойств, а не их наименований, а кнопка «Показать названия» переключает этот режим назад.

Навигация по полям-свойствам отображает в поле «Пояснения к свойству {КОД}» текст, поясняющий суть текущего свойства.

Каждое значение поля-свойства может меняться при помощи ввода текста с клавиатуры. Иногда возможен выбор наиболее типичных значений при помощи выпадающего списка или LOV (кнопки справа от поля или клавиша F9).

Серый фон поля-свойства говорит о том, что у всех выделенных элементов своего значения этого свойства нет, и оно наследуется из родительской конфигурации (при наличии) или используется значение свойства по умолчанию (если редактируется форма в конфигурации DEFAULT).

Если значение поля отображается жирным бордовым цветом, то хотя бы для одного выделенного элемента имеются варианты значений на разных языках. Редактирование такого поля производится посредством локального действия «Многоязычный редактор».

Для полей-свойств доступны следующие локальные действия, отображаемые по клику правой кнопкой мыши в соответствующем поле:

- ***Базовое значение*** – Применимо к полям с белым фоном. Удаляет текущее значение свойства и применяет значение свойства по умолчанию. Поле при этом становится серым.
- ***Вернуть*** – Возвращает перечень всех предыдущих значений текущего свойства элемента формы. Используются данные аудита изменений. Выбор значения из этого перечня помещает его в поле-свойство (возврат старого значения).
- ***Добавить ссылку*** – Отображает список входных параметров, столбцов (для Списков), полей (для Бланков) и некоторых системных переменных формы. Выбор из этого списка помещает ссылку на выбранный элемент в конец текущего значения свойства.
- ***Многоязычный редактор*** – Позволяет задать разные варианты значений для разных языков. В этом случае свойство станет многоязычным. Удаление значений не текущего языка делает значение моноязычным. Это действие также вызывается клавишей Enter, если значение свойство многоязычно.
- ***Скрипт*** – Отображает в отдельном окне скрипт для замены в метаданных текущего значения свойства в текущей конфигурации.
- ***В многострочное окно*** – Открывает бланк для редактирования значения свойства в многострочном поле. Редактирование многоязычных полей производится при помощи действия «Многоязычный редактор». Это действие также вызывается клавишей Enter, если значение свойство моноязычно.

Кнопка «Вверх» перемещает выделенные элементы слева на одну ступень вверх по иерархии. Кнопка «Вниз» делает то же самое в обратную сторону.

Нажатие на кнопку «Сохранить» помещает текущие изменения в хранилище метаданных. Если текущих изменений нет, то кнопка «Сохранить» будет неактивна. Уход с текущего элемента в иерархии редактора метаданных или выделение/развыделение элементов вызывает автосохранение сделанных изменений.

При наличии текущих изменений также доступна кнопка «Вернуть». Её нажатие отменяет все не сохранённые изменения свойств формы.

Программное создание интерфейса

Элементы форм могут полностью или частично создаваться программным способом. Процедуры по созданию элементов интерфейса также могут использоваться для изменения всех или некоторых свойств элементов. Важно, чтобы программные манипуляции с элементами и их свойствами завершились до начала отображения формы в интерфейсе.

Чаще всего используется комбинированный способ загрузки форм: сначала применяются метаданные, а потом, в процессе загрузки, выполняется программный код из свойства ON_LOAD формы (списка или бланка). Этот код может добавить в список столбцы или в бланк поля или в любую форму действия, набор которых не может быть определён в момент разработки, а возникает посредством обращения к тем или иным ресурсам в момент загрузки формы перед её выполнением.

Однако формы полностью порождаться чисто программным способом не могут. Для любой, чисто программной формы, должна существовать пустая форма-заготовка, вызывающая процедуры создания своих элементов в свойстве ON_LOAD (Процедура при загрузке …)

★ Процедуры добавления элементов форм

- **JavaScript**

```
Elements.КодЭлемента={EType:"ТипЭлемента",КодСвойства:"Значение",…,КодСвойства:"Значение"};
```

КодЭлемента – код нового или уже существующего элемента формы;

ТипЭлемента – ITEM | FIELD | ACTION | PARAM | GROUP … Для создаваемого элемента указание типа обязательно. Для существующего – не обязательно.

КодСвойства – задание значений необходимых свойств. Если какое-то свойство не указывать, то для создаваемых элементов будет использоваться значение свойства по умолчанию, а для существующих элементов – оставаться прежние значения неуказанных свойств. Помимо свойств, отображаемых в редакторе, при создании элемента можно использовать свойство Seq – номер элемента по порядку в своём уровне и Parent – код родительского элемента (Parent может быть динамическим, например {FormProperty:Var CurrentField})

- **PL/pgSQL**

Добавление нового элемента формы

```
execute Lui_r_api_set_element_property('КодЭлемента', 'EType', 'ТипЭлемента');
execute Lui_r_api_set_element_property('КодЭлемента', 'КодСвойства', 'Значение');
```

…

Изменение свойств существующего элемента формы

```
execute Lui_r_api_set_element_property('КодЭлемента', 'КодСвойства', 'Значение');
```

…

КодЭлемента – код нового или уже существующего элемента формы;

ТипЭлемента – ITEM | FIELD | ACTION | PARAM | GROUP …

КодСвойства – задание значений необходимых свойств. Если какое-то свойство не указывать, то для создаваемых элементов будет использоваться значение свойства по умолчанию, а для существующих элементов – оставаться прежние значения неуказанных свойств. Помимо свойств, отображаемых в редакторе, при создании элемента можно использовать свойство Seq – номер элемента по порядку в своём уровне и Parent – код родительского элемента (Parent может быть динамическим, например {FormProperty:Var CurrentField})

Управление версиями форм

Режимы управления версиями

Когда пользователь среды разработки приложений создаёт прикладную систему он указывает режим управления версиями форм. Указанное значение сохраняется в БД в параметре FVCS_MODE создаваемой прикладной системы. Возможные значения: N-управление версиями отсутствует; S-одна форма одновременно может модифицироваться только одним разработчиком; M- одна форма одновременно может модифицироваться несколькими разработчиками.

Рекомендуемое значение – M. В этом режиме разработчик формы может пригласить к разработке другого разработчика. Приглашавший и приглашаемый становятся равными в правах на редактирование элементов формы и её конфигураций. Далее любой из них может пригласить к разработке других разработчиков и т.д. Любой участник разработки формы может завершить свою работу с формой. Когда последний разработчик завершает работу с формой, копия её текущей рабочей версии полностью сохраняется в БД, получая очередной номер версии. После этого форма становится доступной для захвата любым разработчиком прикладной системы.

Режим “S” во всём аналогичен режиму “M”, за исключением возможности приглашений к разработке других разработчиков.

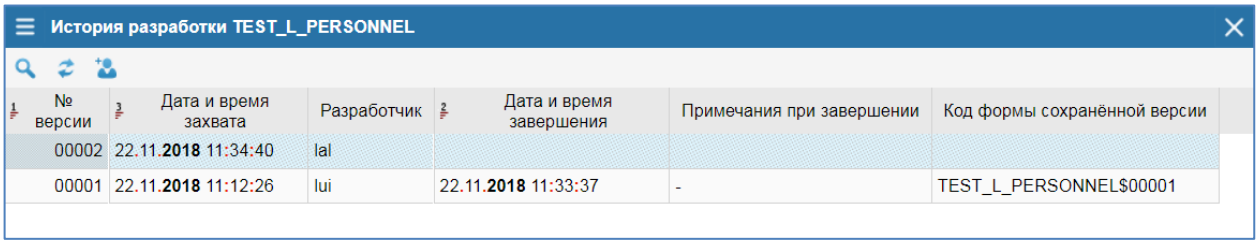
В древовидном списке форм, формы доступные для захвата текущим пользователем помечены иконкой  . Формы, в разработке которых текущий пользователь принимает участие, помечены иконкой  . Формы захваченные другими разработчиками помечены иконкой  .

Захват формы для редактирования

При создании формы она становится “захваченной” тем разработчиком, который её создал. Как только свободная для захвата форма начинает модифицироваться разработчиком, она становится захваченной этим разработчиком. Попытка начать исправления в “чужой” форме вызовет сообщение: LUI-00047.

Пример: LUI-00047. Форма "TEST" приложения "DEMO" удерживается test_dev 12.08.2019 13:12:10. Вам изменения запрещены. Совет: Обратитесь к разработчику с просьбой позволить вам редактировать форму "TEST".

Посмотреть список разработчиков, работающих с формой можно выполнив действие “История разработки формы” в навигаторе редактора форм.



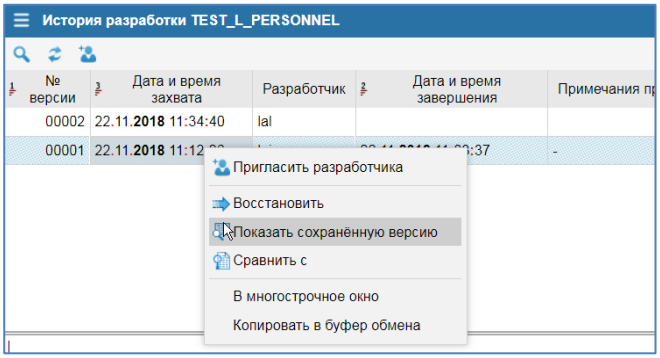
№ версии	Дата и время захвата	Разработчик	Дата и время завершения	Примечания при завершении	Код формы сохранённой версии
00002	22.11.2018 11:34:40	lal			
00001	22.11.2018 11:12:26	lui	22.11.2018 11:33:37	-	TEST_L_PERSONNEL$00001

Сохранение версии формы

Для сохранения текущей версии формы в БД сохранённых версий необходимо выполнить действие “Завершить работу над формой” в навигаторе редактора форм. При этом необходимо будет ввести примечание к сохраняемой версии формы.

Просмотр старой версии формы

В списке “История разработки формы” необходимо установить курсор на нужную версию и выполнить действие “Показать сохранённую версию”.



Редактирование сохранённой версии запрещено.

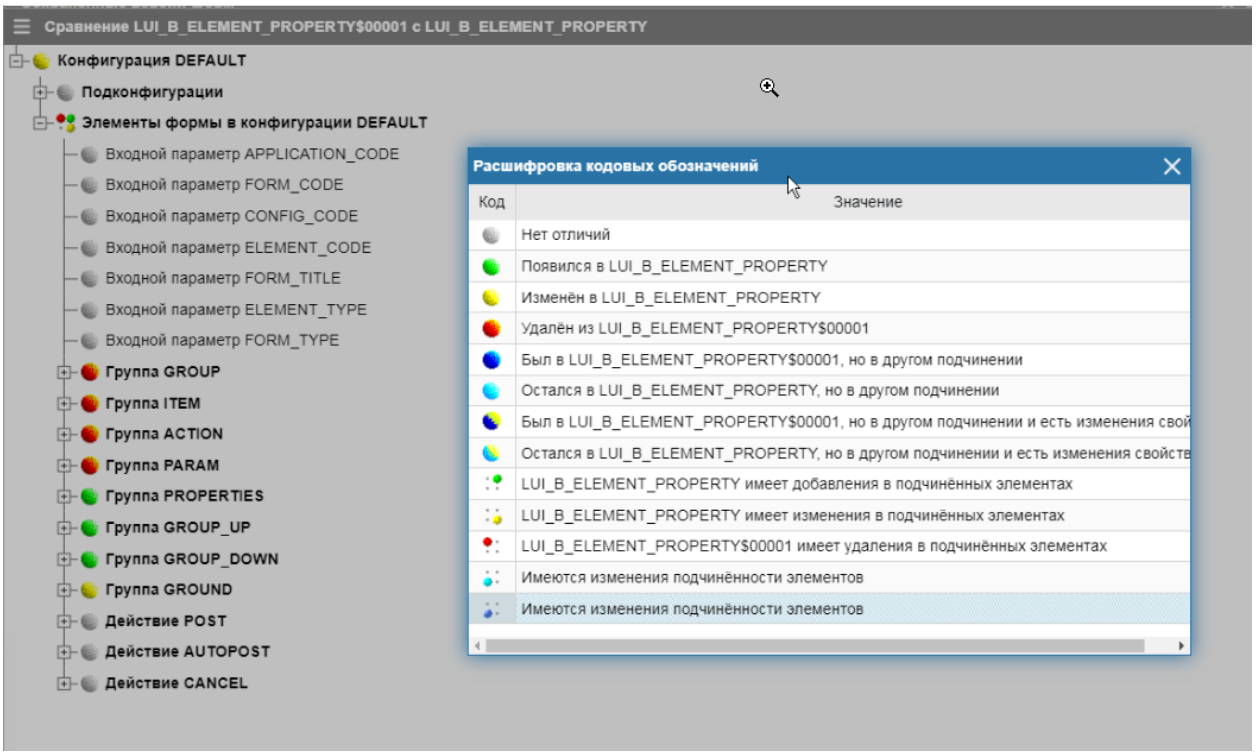
Восстановление старой версии формы

Для восстановления старой версии формы в текущую рабочую версию, необходимо установить курсор на нужную версию и выполнить действие “Восстановить”.

Сравнение версий формы

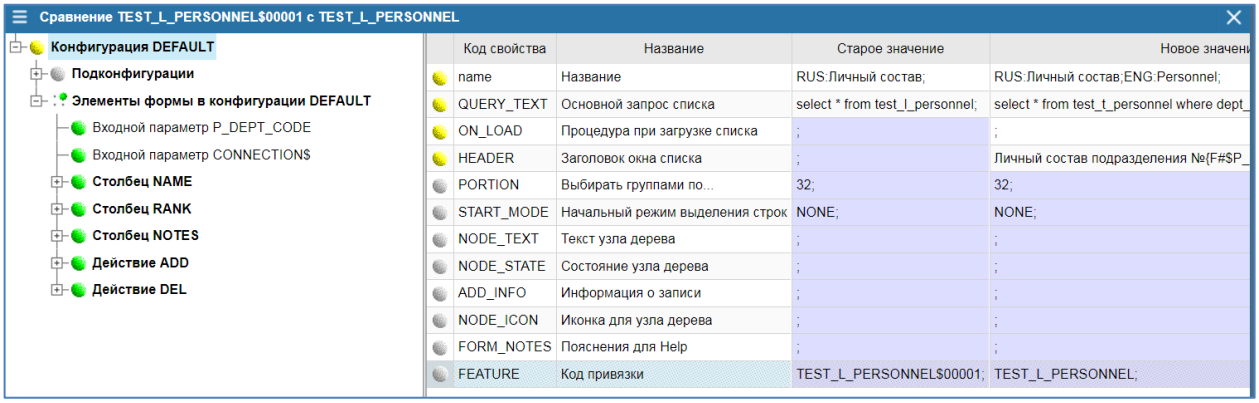
Для сравнения старой версии формы с более модой версией, необходимо установить курсор на нужную версию и выполнить действие “Сравнить с”. Будет отображён список более молодых версий, включая текущую версию. Необходимо выбрать версию для сравнения. Будет отображён иерархический список сравнения версий. В иерархии представлены все элементы обеих версий. Элементы снабжены иконками со смысловой нагрузкой.

Пример:



При навигации на какой-либо элемент, в правой части будет отображён бланк сравнений свойств элемента из разных версий формы.

Пример:



Список всех редактируемых форм прикладной системы

В навигаторе редактора форм имеется пункт “Формы в разработке”. Этот пункт вызывает список всех редактируемых форм. В этом списке могут выполняться следующие действия (вызываются по правой кнопке мыши или иконкой в панели инструментов):

- “История разработки формы” – открывает список версий формы
- “Открыть редактор формы” – вызывает редактор для выбранной формы
- “Скрипт” – вызывает генератор скрипта для переноса формы в другую БД. Есть возможность применять для нескольких отмеченных форм.
- “Завершить разработку формы” – выполняет завершение работы над формой (доступно только для форм, в разработке которых участвует текущий пользователь) Есть возможность применять для нескольких отмеченных форм.
- “Отобразить формы, редактируемые мною” – в списке останутся формы, редактируемые текущим пользователем

Три последних действия позволяют разработчику эффективно выполнять мероприятия по подготовке выпуска очередной версии прикладной системы.

Список всех сохранённых версий форм прикладной системы

В навигаторе редактора форм имеется пункт “Сохранённые версии форм”. Этот пункт вызывает список всех сохранённых версий всех форм приложения. В этом списке могут выполняться следующие действия (вызываются по правой кнопке мыши или иконкой в панели инструментов):

- “Показать сохранённую версию” – открывает редактор только для отображения сохранённой версии
- “Восстановить” – восстанавливает сохранённую версию в текущую рабочую версию
- “Удалить” – удаляет сохранённую версию. Есть возможность применять для нескольких отмеченных версий.
- “Сравнить с” – сравнение версии формы с более молодой версией
- “Скрипт” – вызывает генератор скрипта для версии формы переноса формы в другую БД в качестве текущей версии. Есть возможность применять для нескольких отмеченных версий.
- “Отобрать последние версии ” – в списке останутся самые последние сохранённые версии форм

Два последних действия позволяют разработчику эффективно выполнять мероприятия по подготовке выпуска очередной версии прикладной системы.

Обработка ошибок

LUI предоставляет разработчику приложений развитые средства обработки ошибок и выдачи сообщений на экран браузера и в журнал сеанса.

Справочник сообщений

Справочник сообщений вызывается из главного меню в пункте “Настройки-Справочник сообщений”.

Основные свойства сообщения:

- **Код** – уникальный в пределах прикладной системы код. Настоятельно рекомендуется применять следующую структуру кода: <Префикс>-<Сквозной порядковый номер>. Префикс должен состоять из 2-4 латинских букв и обозначать либо прикладную систему, либо её часть (подсистему, модуль и т.п.). Желательно, чтобы номер представлял собой целое положительное число дополненное нулями слева до пяти разрядов. Под эту структуру разработана форма ввода сообщения в справочник.
- **Тип** – тип сообщения. Возможные значения: E- ошибка, W-предупреждение, I-информация. Сообщение типа “E” это ошибки, прерывающие нормальный ход выполнения формы и действий в формах. Возникновение ошибок вызывает rollback в БД текущего соединения формы. Сообщение типа “W” это предупреждения, говорящие пользователю о том, что действие выполнено, но надо обратить внимание на то или иное обстоятельство. Сообщение типа “I” это информация об успешном выполнении действия.
- **Текст сообщения** – Это текст, выводимый на экран пользователя. Вводится на всех языках приложения. Текст может содержать параметры, которые передаются генератору сообщений из контекста прикладной системы. Параметры кодируются следующим образом: <p1> - первый параметр, <p2>-второй, <p2>-третий,…<p10> -десятый.
- **Рекомендации пользователю** – Это совет пользователю, выводимый на экран. Текст совета вводится на всех языках приложения и параметризирован аналогично тексту сообщения.
- **Перехват ошибки SQL** – это фрагмент, который может встретиться в тексте сообщения, генерируемого СУБД. И если ошибка случилась на уровне СУБД, и этот фрагмент встретился в теле сообщения СУБД, то выполняется перехват сообщения СУБД и подмена его сообщением прикладной системы. Механизм перехвата позволяет сделать сообщения о нарушении ограничений целостности в СУБД понятными для конечного пользователя.

Когда в ходе работы прикладной системы генерируется сообщение, оно разыскивается по коду в справочнике сообщений данной прикладной системы. Если оно не будет найдено, то поиск будет повторён среди сообщений прикладной системы LUI.

Обработка ошибок в сервере приложений

В случае возникновения непредвиденных ошибок в программном коде SQL и PL/pgSQL выдаётся сообщение LUI-00034. В случае возникновения непредвиденных ошибок на уровне JavaScript выдаётся сообщение LUI-00033. В метаязыке LUI имеется команда

ShowMessage:*КодСообщения*;p1:*Значение1;*p2:*Значение2*;…;p10:*Значение10*

Эта команда генерирует сообщение для выдачи его на экран пользователя.

В программе на JavaScript можно вызвать функцию App.msgWriter.write(<код сообщения>,<p1>…<p10>) для генерации сообщения и выдачи его на экран пользователя.

Обработка ошибок на SQL и PL/pgSQL

В схеме базы данных LUI предусмотрена функция lui_r_show_message(<код сообщения>,<p1>…<p10>) для генерации сообщения и выдачи его на экран пользователя.

Пример:

```
{PLPGSQL:(LUI$)
begin
If {Property:Value}>1000 or {Property:Value}<100 then
```

```
 execute lui_r_show_message(‘MSG-00010’);
End if;
Perform set_config('LUI.RETVAL','{Property:Value}',false);

end;}
```

Управление пользователями и правами доступа

В данном разделе описаны формы для управления пользователями и правами доступа в среде разработки приложений. Создаваемые приложения тоже могут содержать формы для управления пользователями и правами доступа. Обычно приложение создаётся с включённой опцией “управление правами”. Разработчик приложения может создать собственные формы на основе имеющихся в LUI, учитывающие специфику прикладной системы.

Пользователи LUI

Среда разработки прикладных систем это тоже прикладная система. И у неё есть средства управления пользователями. Пользователь LUI это пользователь БД, в которой установлена схема LUI и её объекты – таблицы, функции, триггеры и т.п. Список пользователей с функциями создания, удаления и назначением прав вызывается из главного меню “Администрирование-Разграничение Доступа-Пользователи”. При регистрации нового пользователя можно выбрать пользователя уже существующего в БД, где установлена схема LUI. Если пользователя в БД нет, то он будет там создан. При удалении пользователя из LUI, можно выбрать опцию следует ли его сохранить в БД или удалить из числа пользователей БД тоже.

Среди пользователей LUI есть следующие предопределённые пользователи:

- Владелец схемы LUI – главный пользователь, наделённый всеми правами в среде разработки. Удалению не подлежит. Его имя и пароль настраиваются в ходе процесса установки LUI. Как правило, имя – “lui”. Настоятельно рекомендуется не использовать этого пользователя для входа в LUI разработчиков прикладных систем.
- Пользователь/роль коммунальных соединений для доступа к метаданным. Когда сервер приложений выполняет формы прикладных систем, они считываются из схемы LUI. Для этого существует мультиплексируемое прикладными системами коммунальное соединение к БД. Это соединение устанавливается именем этого пользователя. Его имя и пароль настраиваются в ходе установки LUI. Как правило, имя – “lui”.

История событий в отношении пользователя (создание, вхождения пользователя в группы, смена пароля) доступна из главного меню “Администрирование-Разграничение Доступа-История пользователей”.

Группы пользователей LUI

У прикладной системы LUI (т.е. у среды разработки приложений) существуют следующие группы пользователей:

- COMMON – общая группа, все пользователи LUI являются её членами. Соответствует роли коммунальных соединений (lui_common)
- ADMINISTRATOR – группа пользователей, которые могут управлять пользователями и правами доступа
- DEISIGNER - группа пользователей, которые могут создавать и удалять прикладные системы, а также выполнять разработку форм в созданных ими прикладных системах. Кроме того, они могут добавлять пользователей в число разработчиков созданных ими прикладных систем
- DEVELOPER - группа пользователей, которые могут выполнять разработку форм в прикладных системах назначенных для них пользователем ADMINISTRATOR или DESIGNER

Список групп доступен из главного меню “Администрирование-Разграничение Доступа-Группы пользователей”. Для группы пользователей указываются:

- Уникальный код группы в пределах прикладной системы
- Названия на всех языках прикладной системы
- Действие при включении пользователя в группу
- Действие при исключении пользователя из группы

Смысл двух последних свойств состоит в следующем. При включении пользователя в группу или исключении из группы могут потребоваться дополнительные действия в БД. Например, для выполнения операторов

grant и revoke или для вызова серверной процедуры. В этом случае можно применить просто написание нужного оператора SQL в эти свойства. При этом можно ссылаться на псевдопеременные:

- {USER} – имя пользователя
- {USERGROUP} – имя группы пользователей

Пример: grant {USERGROUP} to {USER}

Возможны более сложные случаи, когда при включении в группу или исключении из группы необходимо проделать манипуляции в нескольких БД и даже не только в БД. Для таких случаев в эти атрибуты можно вносить несколько команд на метаязыке LUI. Обрабатываться эти команды должны асинхронной задачей. В составе LUI есть пример такой задачи: LUI_T_USRGRP – Обработка включений пользователей в группы. Эта задача считывает данные из истории действий в отношении пользователя, находит в этой истории данные с необработанными командами и выполняет эти команды. В случае возникновения ошибок они фиксируются также в истории пользователя. Команды могут ссылаться на столбцы извлекаемые основным запросом задачи. Ссылки оформляются в следующем виде:

- {QC:user_name}
- {QC:usergroup_code}

Пример последовательности команд:

DBConnect: MY_CONN2DB1;….

DBConnect: MY_CONN2DB2;….

SQL:(MY_CONN2DB1) grant {QC:usergroup_code} to {QC:user_name}

SQL:(MY_CONN2DB2) grant {QC:usergroup_code} to {QC:user_name}

Здесь “MY_CONN2DB1” и “MY_CONN2DB2” коды соединений с разными БД. Синтаксис DBConnect см. в ***DBConnect – подключение к базе данных.***

Объекты управления правами

Объектами, в отношении которых выполняется назначение прав группам пользователей, являются:

- USER – пользователь. Возможные действия для выдачи или запрета группам:
  - ADD – добавление
  - ADM – администрирование (включение группы и исключение из групп)
  - DEL – удаление
  - UPD – изменение
- APPLICATION – прикладная система. Возможные действия для выдачи или запрета группам:
  - ADD – добавление
  - ADM – администрирование (управление правами на прикладную систему)
  - DEL – удаление
  - UPD – изменение
  - RUN – выполнение
- TASK – задача. Возможные действия для выдачи или запрета группам:
  - ADD – добавление
  - DEL – удаление
  - UPD – изменение
  - RUN – выполнение
- FILES – тип ассоциированных файлов. Возможные действия для выдачи или запрета группам:
  - ADD – добавление файла
  - DEL – удаление
  - UPD – изменение
  - READ – просмотр содержимого файла
- USERGROUP – группа пользователей. Возможные действия для выдачи или запрета группам:
  - ADD – добавление
  - ADM – администрирование (включение пользователей в состав группы и исключение)

    - DEL – удаление
    - UPD – изменение
- UI_ELEMENT – элемент форм. Возможные действия для выдачи или запрета группам:
    - READ – возможность просмотра содержимого поля или столбца
    - RUN – выполнение действий
    - UPD – изменение содержимого поля бланка или ячейки списка

Для каждого из типов объектов устанавливается состояние прав “по умолчанию”. Это позволяет администратору выбрать оптимальную стратегию управления правами. Вот две противоположные стратегии:

- Всё разрешить – потом запрещать для отдельных групп
- Всё запретить – потом разрешать для отдельных групп

В каждой прикладной системе могут быть свои особенные объекты иных типов, в отношении которых проверяются и устанавливаются права доступа. В схеме LUI имеется функция проверки прав доступа на уровне СУБД lui_f_get_effective_right. Она возвращает ‘Y’ если право есть, и ‘N’ – если нет.

Параметры:

- p_obj_type – тип объекта права
- p_obj – имя(код) объекта
- p_act_type – код действия
- p_user – пользователь в отношении которого проверяется наличие права (по умолчанию текущий пользователь прикладной системы)
- p_app_code – код прикладной системы (по умолчанию текущая прикладная система)
- p_show_msg – ‘Y’ –генерировать сообщение об ошибке в случае отсутствия права, ‘N’ – не генерировать (по умолчанию – ‘N’)

Кроме этого, в метаязыке LUI имеется оператор ***Access – наличие права.***

Права группы пользователей LUI

Список назначений и отмен прав для групп доступен по действию “права группы” в списке групп пользователей. При назначении права указывается:

- Тип объекта права - один из типов объектов прав.
- Объект права (допускается ввод в стиле оператора сравнения Like языка SQL). Пример “%”- все объекты, “U%’ – все объекты, начинающиеся с “U”.
- Действие – какое действие надо запретить или разрешить
- Состояние – назначено или отобрано

Управление сеансами пользователей LUI

Сеанс пользователя в прикладной системе не тождественен сеансу в БД. Один сеанс в прикладной системе может открыть много сеансов к разным прикладным БД, или не соединяться с прикладными БД вовсе.

Список сеансов пользователей прикладных систем доступен в главном меню по пункту “Мониторинг-Сеансы в <Код прикладной системы>. В прикладной системе с кодом LUI (т.е. в среде создания прикладных систем) отображаются сеансы пользователей всех прикладных систем. В прочих прикладных системах – только собственные сеансы. Форма списка сеансов позволяет:

- удалить сеанс - работа пользователя будет прекращена, и данные о сеансе (возможно с его журналом отладки) будут удалены.
- прекратить сеанс - работа пользователя будет прекращена, и данные о сеансе (возможно с его журналом отладки) будут сохранены.
- Послать сообщение пользователю сеанса

Все эти действия могут быть выполнены для группы отмеченных сеансов.

Отладка форм

В состав LUI включены следующие средства отладки.

- Выдача дампа памяти сеанса работы в прикладной системе
- Сбор и просмотр журнала действий в режиме отладки

Дамп памяти вызывается действием “Дамп приложения” системного меню. Управление режимом отладки выполняется действиями “Включить/Выключить режим отладки” системного меню. Состоянием “по умолчанию” управляет параметр настройки прикладной системы DEBUG_MODE.

Дамп памяти

Дамп памяти открывается в отдельной вкладке браузера. Дамп может быть использован разработчиком для анализа значений любых свойств соединений с БД, приложений, форм и их элементов. В случае ошибок в прикладной системе у конечного пользователя можно попросить прислать дамп. Страница с дампом сохраняется в файл нажатием Ctrl+s.

Дамп состоит из следующих иерархических разделов:

- Данные о прикладной системе
  - Данные о имеющихся соединениях с БД
  - Свойства и переменные уровня приложения
- Данные о формах, загруженных в память сеанса пользователя сервера приложений
  - Входные параметры формы
  - Свойства и переменные уровня формы
  - Элементы формы подчинённые форме
    - Свойства и переменные уровня элемента формы
      - Элементы формы подчинённые данному элементу
        - Свойства и переменные элемента

Данные о соединениях представлены в виде таблицы со столбцами:

- Код соединения
- Название
- Режим
- Имя пользователя
- URL
- Схема (search path для PostgreSQL)
- Состояние:
  - Valid/Disconnect
  - Transactions – наличие не сохранённых изменений
  - Last activity – дата и время последнего обращения к БД
  - Opened cursors – количество открытых курсоров

Данные о состоянии свойств и переменных уровня приложения, формы и элемента представлены в виде таблицы со столбцами:

- Свойство (код свойства)
- Исходное значение
- Текущее значение
- Размер в памяти
- Количество обращений к значению свойства
- Количество вычислений значения свойства
- Время, измеряемое в миллисекундах, затраченное на последнее вычисление значения свойства
- Зависимости: перечень свойств, значения которых зависят от значения данного свойства и перечень свойств, от значений которых зависит значение данного свойства.

Режим отладки

В режиме отладки, сервер приложений ведёт журнал всех действий выполняемых прикладной системой в рамках сеанса пользователя. Этот журнал доступен для просмотра самим пользователем в пункте системного меню “Просмотр событий”. Чтобы получить журнал событий в виде файла следует войти в журнал событий и воспользоваться пунктом “Версия для печати”. Кроме того, журнал событий доступен в списке сеансов пользователей прикладной системы. Список сеансов вызывается из пункта главного меню “Мониторинг-Сеансы в <Код приложения>”

Дополнения форм

LUI позволяет создавать т.н. “дополнения форм”. Дополнение состоит из формы, разработанной в LUI и действия, которое её вызывают. Предполагается, что разработчик прикладной системы создаёт форму реализующую дополнение, которая может выполнять некие универсальные полезные функции для всех или большинства прочих форм прикладной системы. Например, предоставляет возможность отправить содержимое поля или нескольких полей по электронной почте, или позволяет присоединять к записи любого списка некую дополнительную информацию и т.д. Полезность и необходимость применения дополнений к тем или иным формам прикладной системы определяется не разработчиком, а конечным пользователем. Связь прикладных форм с формой-дополнением устанавливается конечным пользователем. Это делается в пункте меню каждой прикладной системы “Настройки-Дополнения форм”.

В поставку LUI входит пример такого дополнения. Реализует его форма LUI_L_LINKED_FILE. Эта форма позволяет привязать к прикладным данным внешние файлы. Можно загружать их из компьютеров пользователей через браузер в БД LUI, удалять, заменять, отображать их содержимое в браузере. Такие файлы называются ассоциируемыми файлами.

Ассоциируемые файлы

Для того, чтобы воспользоваться функцией ассоциируемых файлов. Пользователь прикладной системы должен описать типы файлов, которые он хочет привязывать к экземплярам сущностей прикладной системы. Это делается в пункте меню прикладной системы “Настройки-Типы ассоциируемых файлов ”.

Необходимо задать код типа файлов, название и иконку.

Примеры:

Код=CONTRACTS, Название=Файл договора, Иконка=contract

Код=USERPHOTO, Название=Фото пользователя, Иконка=user

Код=USERPASS,Название=Скан паспорта пользователя, Иконка=pass

Регистрация дополнений

В прикладной системе надо зарегистрировать дополнение. Основные свойства дополнения:

- Код – уникальный код дополнения
- Название – название дополнения на всех языках приложения
- Описание – подробное описание назначения дополнения
- Иконка – имя иконки, которая будет символизировать данное дополнение в прикладной системе
- Тип, Код и конфигурация формы реализующей дополнение (т.е. форма, которая будет вызываться из прикладных форм)
- Запрос – оператор SELECT. В это поле надо ввести запрос, который позволит найти строку в какой-то таблице, к которой привязано дополнение. Запрос должен извлекать одну строку, один столбец в котором представлен какой-то прикладной идентификатор. Условие поиска должно быть записано в виде where col1={F:KEY1} [...and colN={F:KEY2}. Нумерация полей ключа выполнена сортировкой имён ключевых столбцов по алфавиту. Этот запрос позволит выявлять лишние файлы, образовавшиеся в результате удалений данных из таблиц приложения. Примеры: select contract_no from contracts where id={F:KEY1} select subject from meetings where meeting_date='{F:KEY1}'::timestamp and meeting_place='{F:KEY2}'

Для дополнения реализующего ассоциируемые файлы необходимо ввести список типов файлов, которыми может оперировать данное дополнение.

Для каждого дополнения необходимо ввести список форм, из которых должен быть доступен вызов формы дополнения. При этом указывается:

- Код и конфигурация вызывающей формы
- Ключ - Набор полей или входных параметров вызывающей формы, совокупность значений которых однозначно идентифицируют строку в таблице из поля “Запрос” формы регистрации дополнения. Ключ задаётся в массива JSON. Пример: ["P_APPCODE","P_USER_NAME"]

- Заголовок дополнения – Заголовок окна вызываемой формы. Ссылка вида {F:P_ADDONNAME} позволяет применить название дополнения, введённое в поле “Название” формы регистрации дополнения.
- Необходимость автоматического вызова (режим мастер-деталь)
- Уровень, на котором в вызываемой форме будет размещаться действие для вызова формы дополнения