DCL_Описание операций над объектами

Описание операций над объектами

Операции над объектами комплексного типа

field$–операции и ссылки – перебор полей записи

Набор field$-операций и ссылок позволяет обращаться к полям записи комплексного объекта по их порядковому номеру путем перебора. Реализованы следующие операции позиционирования поля записи:

- field$first – позиционировать первое поле
- field$next – позиционировать следующее поле

Данные позиционированного поля доступны по следующим ссылкам:

- field$name – имя поля
- field$value – значение поля

В том случае, если при позиционировании происходит выход за область полей записи, ссылка field$name содержит пустое значение.

Пример

```
for(i=0 ; i< v_data.count ; i++) {


 for(j=0 ; ; j=1) {
                if(j==0)  v_data[i].field$first ;
                else      v_data[i].field$next ;


                if(sizeof(v_data[i].field$name)==0)  break ;
```

```
        v_fld_name <==v_data[i].field$name ;
        v_fld_value<==v_data[i].field$value ;
                                                    ...
    }
            }
```

---

clear – удаление всех записей

Производит удаление всех записей объекта.

Ничего не возвращает.

Пример

```
Xobject.clear
```

---

add – добавление новой записи

Производит добавление новой пустой записи к объекту. Добавленная запись становиться текущей записью комплексного объекта

Ничего не возвращает.

Пример

```
Xobject.add
```

---

delete – удаление записи

> К сведению
> Раздел находится в разработке

---

count – число записей объекта

Возвращает число записей в составе объекта.

Пример

```
cnt=Xobject.count
```

---

add_field – добавление поля в структуру записей

Производит добавление поля заданного типа ко всем записям комплексного объекта. Добавляемое поле описывается символьной строкой в соответствии с базовым синтаксисом описания полей комплексных объектов.

Ничего не возвращает.

Пример

```
Xobject.add_field("char order[20]")
```

---

tracesize – разрешить/запретить следящую коррекцию элементов комплексного объекта

Разрешае/запрещает реализацию режима следящей коррекции для значений элементов заданного комплексного объекта.

Параметры:

- Флаг разрешения следящей коррекции: 1 - разрешить, 0 - запретить

По-умолчанию следящая коррекция запрещена.

Синтаксис

object.tracesize(<0 или 1>)

find – поиск записи по условию или AND-набору условий

Производит поиск по заданному условию среди записей комплексного объекта. Условие поиска задается в виде набора пар параметров, где первый параметр пары задает имя поля, а второй – его значение. Критерии, задаваемые отдельными парами, объединяются логической операцией AND. Если условие поиска не задано, то каждая запись полагается соответствующей условию.

При выполнении оператора производится разметка всех записей, соответствующих условию для обеспечения последующего применения операторов findnext и findprev.

Может использоваться совместно с режимом «MARKED_ONLY».

При наличии записей, соответствующих условию, первая из них становиться текущей записью комплексного объекта.

Возвращает число записей, соответствующих условию.

Пример

cnt=Xobject.find("branch", 0, "type", "TABLE")

---

sort – сортировка записей по заданному полю

Производит сортировку записей по возрастанию значения заданного поля.

Синтаксис

object.sort(<Имя поля сортировки>)

Записи не содержащие заданного поля смещаются «вниз».

Положение текущей записи – не определено.

Пример

Xobject.sort("order")

---

compare – сравнение двух наборов записей

Производит сравнение двух наборов записей двух комплексных объектов с использованием в качестве ключа первого по порядку поля в записях. Сравнение записей производится по их физическому представлению, то есть без учета имен полей. Перед использованием данной операции сравниваемые наборы должны быть упорядочены по возрастанию значения ключевого поля, например, с помощью метода sort.

Синтаксис

object.compare(<объект_1>,<объект_2>,<режим сравнения>)

Допустимы следующие режимы сравнения:

"ANY" или NULL - все различия в наборах записей

"NOT_IN_1" - только записи, имеющиеся во втором объекте и отсутствующие в первом

"NOT_IN_2" - только записи, имеющиеся в первом объекте и отсутствующие во втором

"BY_VALUE" - только записи, имеющиеся в обеих объектах и различающиеся по составу полей кроме ключевого.

Положение текущей записи – не определено.

Пример

Xobject.compare(Object_1, Object_2, "NOT_IN_2")

---

marked_only – управление режимом использования разметки записей

Включает (выключает) режим использования только размеченных записей в качестве объекта операции.

Синтаксис

object.marked_only({0 | 1})

Пример

Xobject.marked_only(0)

---

get_files_by_mask – добавляет записи, содержащие данные о файлах, сответсвующих заданной маске

Добавляет в конец набора записей объекта новые записи специальной структуры, содержащие данные о файлах, соответствующих заданной маске. Структура добавленной записи включает следующие элементы:

- path - путь к файлу
- name - имя файла (без расширения)

«Объявленная» структура записей объекта может отличаться от данной структуры.

Структура маски допускает любое количество «звездочек», при этом обеспечивается проход по поддеревьям.

Пример

Xobject.get_files_by_mask("/sport/data/*")

Xobject.get_files_by_mask("/sport/*/*АВТР*/*.cpp")

---

read_csv – чтение данных из файла по шаблону

Производит чтение из файла данных согласно заданному шаблону и их помещение в структуру объекта комплексного типа.

Синтаксис

object.read_csv(<Путь к файлу>, <Шаблон>)

Шаблон представляет собой последовательность произвольных текстовых фрагментов и названий полей комплексного объекта, причем названия полей ограничены символами %.
В том случае, если значение должно быть проигнорировано в качестве имени поля указывается спецификатор IGNORE (в любом регистре).
Если в имени поля первым полем следует символ "плюс", то значение будет добвалено справа к значению данного поля. Данная возможность может использоваться для сбора единого поля из нескольких отдельных компонент. При этом следует учитывать, что обработка спецификаций полей идет строго слева направо по структуре шаблона.
Если указанное в шаблоне поле отсутствует в структуре объекта - будет выдана ошибка.
Для указания в текстовом фрагменте символа % используется последовательность %%.

Записи должны располагаться в файле построчно. В том случае, если структура записи не соответствует заданному шаблону – она игнорируется.

Возвращает число считанных из файла записей.

Пример считывания файла

```
typedef <data> {
                        char  type[16] ;
```

```
                    char  line[8] ;
                    char  name[32] ;
} ;
<data>  Xobject ;

Xobject.read_csv ("c:/data/1.txt", "Type %type%:%line%:%name%") ;

if(errno!=0) {
                 show("File read error " @ atos(errno)) ;
                    return(-1) ;
}
```

При считывании файла следующего содержания:

```
Type type_1:2:Integer
Type type_2:3:String
Class class_1:4:Class first
Type type_3:5:Long
Type type_4:6=type_1
```

Будут добавлены следующие записи:

```
{"type_1", "2", "Integer"}
{"type_2", "4", "String"}
{"type_3", "5", "Long"}
```

Пример с частичным игнорированием и объединением полей

```
typedef <data> {
                    char  key[16] ;
                    char  remark[32] ;
} ;

<data>  Xobject ;

Xobject.read_csv ("c:/data/1.txt", "%ignore%#%ignore%#%key%#%ignore%#%+key%#%ignore%") ;
```

При считывании файла следующего содержания:

```
1##30109810400000200341#2015-07-08#CLOSE#2015-07-10#
2##40702810100000189572#2015-07-08#OPEN#2015-07-10#
3##40702810370000900234#2015-07-08#CLOSE#2015-07-10#
```

Будут добавлены следующие записи:

```
{"30109810400000200341CLOSE", ""}
{"40702810100000189572OPEN", ""}
{"40702810370000900234CLOSE", ""}
```

write_csv – запись данных в файл по шаблону

Производит запись в файл данных комплексного объекта согласно заданному шаблону или, при отсутствии такового, в соответствии с CSV-схемой.

Синтаксис

```
object.write_csv(<Путь к файлу>[, <Шаблон>])
```

Шаблон представляет собой последовательность произвольных текстовых фрагментов и названий полей комплексного объекта, причем названия полей ограничены символами %. Для указания в текстовом фрагменте символа % используется последовательность %%.

При отсутствии шаблона данные пишутся построчно в порядке следования полей с использованием в качестве разделителя символа «точка с запятой».

Может использоваться совместно с режимом «MARKED_ONLY».

Возвращает число записанных в файл записей.

Пример

```
Xobject.write_csv("c:/data/1.txt")

Xobject.write_csv ("c:/data/1.txt", "%type%:%line%:%name%:%out_str%\n")
```

read_dbf – чтение данных из DBF-файла

Производит чтение из DBF-файла данных согласно заданному списку полей и их помещение в структуру объекта комплексного типа.

Параметры:

- Путь к имени DBF-файла
- Список отбираемых из файла полей. Если список пустой - берутся все поля DBF-файла

Синтаксис

object.read_dbf(<Путь к файлу>, <Список полей>)

Возвращает число считанных из файла записей.

Пример считывания файла

```
<unknown>  soun ;

     soun.clear ;
     soun.read_dbf("c:\\rn-net\\rnagent\\templates\\dbf\\soun1.dbf", "KOD,NAIM,ADRES") ;
  if(soun.count==0) {
                      $error<=="SOUN DBF-file load error : " @ atos(errno)  ;
        signal("ABTP", v_SignalServer, v_SignalName, "ERROR", $error) ;
                                  return(-1) ;
                  }
```

form_xml – формирование записи данных в виде XML-структуры

Выдает данные комплексного объекта в виде XML-потока.

Синтаксис

object.form_xml(<Тэг кадра записи>, <Шаблон структуры записи>)

Тэг кадра записи задает имя тэга, заключающего в себе поля одной записи.

Шаблон структуры записи – задает перечень выводимых полей, соответствующие им тэги и преобразования. Спецификации отдельных полей в шаблоне разделены запятыми: <Field_1>,<Field_2>,…

Спецификация поля имеет следующую структуру: <Тэг>=<Имя поля>[:<Преобразования1>]

Может быть указано более одного спецификатора преобразования – в этом случае они разделяются символом «точка с запятой».

Возможные спецификаторы преобразования:

YMD>DMY - дата вида YYYY.MM.DD преобразуется в DD.MM.YYYY, при этом преобразованию подвергаются только данные длинной не менее 10 символов

CDATA - данные тэга будут залючны в спецификаторы CDATA - <![CDATA[…]]>

NL_IGNORE - в данных тэга символы перевода строки будут заменены на пробел.

Если в шаблоне структуры записи указан спецификатор TAGBYROW, то каждый тэг выводится построчно.

Если шаблон структуры записи не задан или содержит только спецификатор TAGBYROW, то в XML выводятся все поля, при этом в качестве тэга поля используется его название.

Может использоваться совместно с режимом «MARKED_ONLY».

Возвращает текстовую строку с результатом преобразования.

Пример

Например, рассмотрим комплексный объект следующего содержания:

Запись\Поле	A	B	C

1	12345	2013.08.15	Dummy 1
2	6789	None	Dummy 2

Тогда результатом выполнения оператора

xml<==Xobject.form_xml("Frame", "ID=A,Date=B:YMD>DMY")

будет

<Frame><ID>12345</ID><Date>15.08.2013</Date></Frame>

<Frame><ID>6789</ID><Date>None</Date></Frame>

А для оператора

xml<==Xobject.form_xml("Frame", NULL)

результатом будет

<Frame><A>12345</A><B>2013.08.15</B><C>Dummy 1</C></Frame>

<Frame><A>6789</A><B>None</B><C>Dummy 1</C></Frame>

---

sql_select_once – выполнение SELECT-запроса для соединения по-умолчанию

Добавляет в конец набора записей объекта новые записи, полученные в результате выполнения SELECT-запроса.

Синтаксис

object.sql_select_once(<Rows>, <SQL>[, <Variable_1>, …, <Variable_N>])

где,

Rows - максимальное число выбираемых строк

SQL - текст выполняемого SELECT-запроса

Variable_? - переменная, подставляемая при выполнении SELECT-запроса вместо одноименного маркера (например, переменная с именем «v_text» подставляется вместо маркера «:v_text»)

При ошибке возвращает -1 и выставляет значение системной переменной errno. При ошибке E_SQL_ERROR текст ошибки может быть запрошен через процедуру sql_error.

Пример

```
Xobject.sql_select_once(1, "select count(*) from clients");

Xobject.sql_select_once(10, "select * from clients where type=:c_type", c_type);
```

MIDAS_API_new – создание шаблона сообщения MIDAS_API

Удаляет все записи объекта и добавляет запись в соответствии с шаблоном указанного сообщения MIDAS API.

Синтаксис

object.MIDAS_API_new(<MsgCode >, <MsgDesc>),

где

MsgCode - код сообщения

MsgDesc - раздел файлов описания сообщений или прямая спецификация полей сообщения (тогда значение параметра начинается с символа '@')

При использовании задании структуры сообщения через файл описания, каждому используемому сообщению

соответствует свой файл описания, размещаемый в разделе MsgDesc, имеющий имя, совпадающее с кодом сообщения и расширение «csv». Файл описания состоит из набора строк, каждая из которых (кроме первой) должна содержать минимум два слова, разделенные символом «точка с запятой». Первое поле содержит название поля сообщения, второе – длину поля в символах. Часть стоки, следующая за вторым словом игнорируется. Первая строка файла описания игнорируется. Ниже приведен пример файла описания сообщения CUSD:

Имя файла – CUSD.csv

Содержимое:

```
name;size;description;datatype;reference_to;trueValue;falseValue
ADDCNA1;70;Cust. Name and Address 1;string;;;
ADDCNA2;70;Cust. Name and Address 2;string;;;
ADDCNA3;35;Cust. Name and Address 3;string;;;
ADDCNA4;35;Cust. Name and Address 4;string;;;
ADDCSSN;10;Customer Shortname;string;;;
DD1AD1;28;holder 1 address 1;string;;;
...
```

При использовании прямой спецификации полей сообщения строка, описывающая структуру сообщения, начинается с символа '@' и состоит из набора описателей полей, разделенных символом «запятая». Каждый описатель поля представляет собой пару значений – имя поля и его длину в символах, разделенных символом «двоеточие».

Шаблон сообщения представляет собой запись с полным набором полей сообщения (создаются пустыми) и системными полями: API_object со значением 'REQUEST' и API_message со значением равным коду сообщения.

При успешном выполнении возвращает 0, при ошибке - -1.

При ошибке в объект вместо шаблона сообщения добавляется запись об ошибке, содержащая два поля: API_object=«DRIVER_ERROR» и text – с описанием ошибки. Для проверки наличия ошибок может быть использован метод MIDAS_API_errors.

Пример

```
Xobject.MIDAS_API_new("CUSD", "C:/MidasApi/");
Xobject.MIDAS_API_new("CUSD", "@ADDCNA1:70,ADDCSSN:10,DD1AD1:28");
```

MIDAS_API_send – передача сообщения MIDAS_API

Передает сообщение, сформированное на основе шаблона на MIDAS API по указанному интерфейсу.

Синтаксис

object.MIDAS_API_send(<Action>, <Login>, <URL>),

где

Action	-	код операции: VALIDATE, WRITE, UPDATE и так далее
Login	-	пользователь/пароль
URL	-	<Тип интерфейса>://<Адрес получателя>. Допустимые типы интерфейсов: SOAP

Перед отправкой сообщения удаляет все записи объекта, кроме шаблона сообщения. Если и добавляет запись в соответствии с шаблоном указанного сообщения MIDAS API.

Пример

```
Xobject.MIDAS_API_send("WRITE", "user/pass", "SOAP:// s-msk34-ast02…");
```

MIDAS_API_errors – извлечение сообщений об ошибках MIDAS_API

> К сведению
> Раздел находится в разработке

Надо еще добавить функции – «подготовка к UPDATE» и «перевод REPLY в «REQUEST»

QC_result_new – создание шаблона записи QC_result

Добавляет запись, содержащую поля для описания результата теста в Quality Center.

Синтаксис

object.QC_result_new()

Запись содержит следующие поля:

$type$	-	всегда «QC-RESULT»
test_path	-	абсолютный путь к экземпляру теста в QC
result	-	результат выполнения теста: Passed, Failed
bug_desc	-	описание дефекта (только для результата Failed)

Пример

Xobject.QC_result_new();

QC_result_send – осуществляет загрузку в Quality Center записи QC_result

Обрабатывает все записи вида QC_result в соответствии с настройками обмена с Quality Center.

Синтаксис

object.QC_result_send(<Script>, <Folder>, <Project>, <TestId>),

где

Script	-	Шаблон вызова исполняемого файла обработчика
Folder	-	Папка для создания обменных файлов
Project	-	Название проекта
TestId	-	Идентификатор «попытки» (RunName в терминологииQualityCenter)

Шаблон вызова исполняемого файла обработчика представляет собой строку командного интерпретатора, в которой размещены следующие обязательные макроподстановки:

$REQUEST$	-	путь к файлу запроса
$RESULT$	-	путь к файлу результата

Файл запроса представляет собой многострочный текстовой файл следующей структуры:

Project=<Имя проекта>

Test_Id=<Идентификатор «попытки»>

Test_Path=<Абсолютный путь к экземпляру теста>

Result=<Результат теста>

Comment=<Описание дефекта>

Файл результата представляет собой текстовой файл, первая строка которого содержит информацию о результате

обработки данных файла запроса – если строка содержит слово «Success», то запрос считается успешно обработанным, в противном случае – обработанным с ошибкой.

Файл запроса :

Project=QualityCenter_Demo

Test_Id=BP1_20100907_6

Test_Path=Root\ALM System testing\Datamart-KRM Input\CHATR\PORT\[1]HLDYID_PMT

Result=Failed

Comment=Difference in fields values(RBRU_00000021, 34618,87, 34620...

Файл результата :

Success

При ошибке возвращаемое значение не равно 0. Для получения текста ошибки следует использовать метод QC_errors.

Пример

```
Xobject.QC_result_send("D:\QualityCenter\QC_bug_registry.vbs $RESULT$ $REQUEST$", "D:\QualityCenter\Work",
"QualityCenter_Demo",  "BP1_20100907_1");
```

QC_errors – извлечение сообщений об ошибках обработки записей вида QC_result

Возвращает текст сообщения об ошибке после использования метода QC_result_send.

Синтаксис

```
object.QC_errors()
```

Пример

```
error_text<==Xobject.QC_errors();
```

EMAIL_new – создание шаблона email

Добавляет запись, содержащую поля для описания email.

Синтаксис

```
object.EMAIL_new()
```

Запись содержит следующие поля:

$type$	-	всегда «EMAIL»
receivers	-	список получателей, разделитель – запятая или точка с запятой
sender	-	отправитель
subject	-	тема письма
body	-	тело письма
files	-	перечень файлов-вложений, разделитель – запятая или точка с запятой
header	-	дополнительные поля заголовка
error	-	ошибка обработки сообщения

Пример

```
Xobject.EMAIL _new();
```

EMAIL_send – отправка email

Производит отправку email-сообщений в соответствии с выбранным способом и параметрами отправки. Из набора записей комплексного объекта обработке подвергаются только записи, имеющие поля $type$=«EMAIL».

Синтаксис

object.EMAIL_smtp(<Method>, <Control>[, <Folder>])

где,

Method - Используемый метод отправки: SMTP, EXTERNAL

Control - Управляющая информация для отправки

Folder - Папка для размещения рабочих файлов

Метод отправки определяет механизм с помощью которого будет производится отправка сообщений. Поддерживаются следующие методы отправки:

- SMTP - использование встроенного почтового SMTP-клиента
- EXTERNAL - использование внешней утилиты отправки с передачей данных через командную строку
- EXTERNAL@ - использование внешней утилиты отправки с передачей данных через обменные файлы

Для SMTP управляющая информация содержит адрес соединения в формате:

<IP/DNS адрес>:<порт>

Для EXTERNAL управляющая информация представляет собой командную строку, которая может содержать следующие макроподстановки:

%receivers% - список получателей, разделитель – запятая

%sender% - отправитель

%subject% - тема письма

%body% - путь к файлу с телом письма

%files% - перечень файлов-вложений, разделитель – запятая

%header% - дополнительные поля заголовка

Для EXTERNAL@ управляющая информация представляет собой командную строку, которая может содержать следующие макроподстановки:

%send_path% - путь к файлу, содержащему описание сообщения

%done_path% - путь к файлу, содержащему результат отправки

Файл описания сообщения представляет собой текстовой файл следующей структуры:

RECEIVER=<получатель>

SENDER=<отправитель>

SUBJECT=<тема письма>

BODY=<тело письма>

FILE=<путь к файлу вложения>

HEADER=<дополнительное поле заголовка>

SEND MESSAGE

Полей RECEIVER, BODY, FILE и HEADER может быть несколько. Число полей BODY определяется числом строк в теле сообщения.

Файл результата отправки представляет собой текстовой файл, который содержит либо слово «SUCCESS» - при успешной отправке, либо описание ошибке – при неудаче.

При ошибке возвращаемое значение не равно 0. Детальное сообщение об ошибке записывается в поле error каждой из EMAIL-записей. Для получения текста ошибки следует либо использовать метод EMAIL_errors, либо произвести «ручное» сканирование EMAIL-записей.

Пример

```
object.EMAIL_send("SMTP", "smtp.bank.ru:25") ;

object.EMAIL_send("EXTERNAL", "cat %body% | mailx -r \"%sender%\" -s \"%subject%\" %receivers%") ;

object.EMAIL_send("EXTERNAL@", "wscript send.vbs %send_path%  %done_path%") ;
```

EMAIL_errors – извлечение сообщений об ошибках обработки email

Анализирует все EMAIL-записи на предмет наличия записи об ошибки в поле error. При наличии ошибок формирует сводное сообщение, состоящее из последовательно соединенных сообщений об ошибках, относящихся к отдельным EMAIL-записям.

Может применяться для получения информации об ошибках после использования метода EMAIL send.

Синтаксис

```
object.EMAIL_errors()
```

Пример

```
error_text<==Xobject.EMAIL_errors();
```

#