Введение в DaqConfig.

Оглавление

Аннотация

Программный пакет **CRW-DAQ** [1,2,3] имеет несколько встроенных специализированных языков, ориентированных на решение разных задач. Например, язык **DaqScript** служит для интерпретации «на лету» поступающих в каналы связи команд, а **DaqPascal** – для создания алгоритмов сбора данных и управления. Этот документ содержит обзор языка **DaqConfig**, созданного для описания параметров пакета **CRW-DAQ** и конфигурации прикладных программ для измерительных систем, работающих под его управлением.

Общий обзор и назначение DaqConfig

Язык **DaqConfig** – простой язык для описания **конфигурации** как самого пакета **CRW-DAQ**, так и прикладных программ, работающих под его управлением. С точки зрения прикладного программиста **DaqConfig** – это набор **правил** и **соглашений**, в согласии с которыми создаются **конфигурационные файлы** для описания параметров и данных прикладных программ, а также **API**, т. е. набор функций (например, **readini**, **wordcount**, **extractword**, **readinisection**) для интерпретации этих файлов.

Что такое **конфигурация** программы? Классическая книга [10] по программированию от создателя языка Pascal Никлауса Вирта называется «**Алгоритмы и структуры данных**». Этим названием автор подчеркивает, что алгоритмы и данные являются двумя основными понятиями программирования. В пакете **CRW-DAQ** языки **DaqPascal** и **DaqScript** являются языками описания **алгоритмов** управления, а язык **DaqConfig** служит для описания **структуры данных**. Именно он задает объекты и структуры данных (теги, кривые, окна, устройства, прикладные программы), а также определяет их начальные значения. Описание структуры данных **ПО** – это и есть его конфигурация.

Язык **DaqConfig** является **декларативным**. Это значит, что он описывает (декларирует) структуры данных, но не предписывает строго, когда и как они будут созданы. Программы создают структуры данных тогда, когда это им необходимо, и освобождают их после использования, когда они больше не нужны. Некоторые структуры данных создаются при загрузке пакета **CRW-DAQ** и существуют до его завершения, другие структуры (например, конфигурации измерительных систем) загружаются и освобождаются по требованию пользователя.

Интерпретатор **DaqConfig** рассматривает исходные файлы как **секционированные текстовые файлы**, в которых **текст** разделяется на поименованные логические фрагменты – **секции**, идентифицируемые при помощи **заголовков секций** в квадратных скобках **[…]**, причем каждая секция может рассматриваться как отдельный поименованный текст. Самым простым примером секционированных текстовых файлов являются ***.ini** файлы в **ОС Windows**. Поэтому общие правила создания файлов конфигураций, принятые в **DaqConfig**, похожи на правила построения ***.ini** файлов, однако со многими дополнениями, которые расширяют возможности по сравнению с обычными ***.ini** файлами.

Интерпретатор **DaqConfig** является **инструментом**, который часто используется во многих модулях пакета для решения разных задач. Он имеет общий **синтаксис** (правила построения выражений), но разные **соглашения** (дополнительные правила интерпретации) в зависимости от **типа файла** (расширения) и **целевого объекта**, который использует интерпретатор **DaqScript** для решения своих задач.

В таблице 1 кратко перечислены основные сферы применения языка **DaqConfig**, разбитые по типам файлов и целевым объектам, в которых он используется.

- В главном файле инициализации **Crw32.ini** (который имеет несколько «вложенных» ***.ini** файлов) задаются параметры и начальные значения переменных ядра пакета **CRW-DAQ**.
- Файлы конфигурации ***.cfg** (от слова **c**on**fig**uration) служат для описания структуры данных, состава оборудования и прикладных программ **DAQ** системы.
- Файлы ***.crc** (от **c**ir**c**uits) содержат описания окон мнемосхем для создания графического интерфейса пользователя **DAQ** системы.
- Файлы ***.cal** (от **cal**ibration) содержат описание калибровок измерительных каналов **DAQ** системы.
- Файлы ***.ini** (от **ini**tialization) служат для хранения параметров и инициализации (задания начальных значений) переменных прикладных программ **DAQ** системы.
- Файлы ***.spd** (от **sp**ectral **d**ata) служат для хранения спектрометрических данных.
- Файлы ***.dat** (от **dat**a) содержат записи данных динамических (зависящих от времени) измерений **DAQ** системы.

Как видим, язык **DaqConfig** имеет много важных применений. Поэтому пакет **CRW-DAQ** сложно представить без этого интерпретатора.

*Таблица 1. Основные сферы применения интерпретатора **DaqConfig**.*

№	Тип	Целевой Объект	Выполняемые функции
1	***.ini**	**Ядро пакета CRW-DAQ** Файл инициализации Crw32.ini	Инициализация переменных пакета **CRW-DAQ**. Задание параметров **runtime** системы **Crw32.exe**. Выполнение стартового скрипта **[System.StartupScript]**.
2	***.cfg**	**DAQ система** Файлы конфигурации оборудования и **РПО**	Описание конфигурации оборудования, структур данных и состава **РПО** для прикладных измерительных систем и систем управления.
3	***.crc**	**DAQ система** Окно мнемосхемы **(Circuit_Window)**	Описание окон мнемосхем (**Circuit_Window**) - графических окон для создания интерактивного интерфейса пользователя в прикладных программах **DAQ** системы.
4	***.cal**	**DAQ система** Файл калибровки канала измерений	Описание калибровки измерительного канала, т. е. математического преобразования для перевода измеренных показаний датчика в физические единицы.
5	***.ini**	**DAQ система** Файлы инициализации	Хранение и загрузка значений параметров и инициализация начальных значений переменных прикладных программ.
6	***.spd**	**DAQ система** Файлы спектров	Хранение и загрузка спектров, т. е. статистических данных измерений в виде спектров (гистограмм).
7	***.dat**	**DAQ система** Файлы данных динамических измерений	Хранение и загрузка данных динамических измерений, когда данные измерений хранятся в виде кривых, в абсциссе которых записано время, а в ординате – данные измерений.
8	***.lmd**	**DAQ система** Файлы данных ядерно-физических измерений	«List Mode Data». Хранение и загрузка данных ядерно-физических измерений с многопараметрическими событиями.

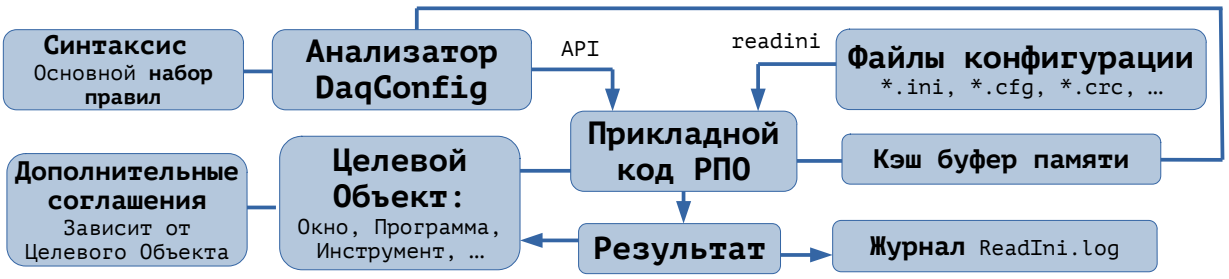

*Рисунок 1. Общая схема работы интерпретатора **DaqConfig**.*

На рисунке 1 представлена общая схема работы интерпретатора **DaqConfig**. Анализатор **DaqConfig** поддерживает **синтаксис**, т. е. основной набор правил языка и предоставляет набор функций **API** для прикладного кода **РПО**. С помощью этого **API** (например, **readini**) и дополнительных соглашений, зависящих от целевого объекта, код **РПО** считывает данные из файлов конфигурации (***.ini**, ***.cfg**, …), а результат чтения и анализа передает целевому объекту.

Кроме того, результат чтения и анализа данных заносится в журнальный файл (**Readini.log**). Это позволяет контролировать работу интерпретатора, что нередко помогает находить ошибки конфигураций.

Для ускорения работы интерпретатора используется кэш буфер памяти, который отслеживает время последнего изменения файлов конфигурации и позволяет избежать повторного чтения и анализа недавно прочитанных файлов. Кэш буфер имеет настройки (через команду **@memory** в окне **Главной Консоли**) и освобождается по мере необходимости, что позволяет балансировать между скоростью доступа и объемом занятой кэш буфером памяти.

Использование кэш буфера позволяет кардинально (в десятки раз) ускорить работу **DaqConfig**. Ускорение достигается за счет того, что в большинстве случаев прикладной код считывает много параметров из одного набора файлов конфигурации, а это позволяет брать содержимое файлов из кэш буфера, избегая повторного чтения файлов. Данные в кэш буфере уже сгруппированы по файлам и секциям и имеют быстрый доступ по именам через хэш таблицы. Поэтому извлечение данных из кэш буфера происходит намного быстрее, чем чтение файла заново. При этом кэш буфер обновляется автоматически, если файл конфигурации изменяется, что достигается за счет наблюдения даты изменения и размера файлов.

Интерпретатор **DaqConfig** создавался специально для разработки систем сбора данных и управления. При его создании большое внимание уделялось обеспечению высокой скорости работы, надежности и устойчивости к ошибкам входных данных.

Синтаксис интерпретатора DaqConfig

Интерпретатор **DaqConfig** имеет простой синтаксис, который можно кратко описать в виде набора правил:

- **Исходный файл** входного текста читается как поток байтовых **строк** (8 бит на символ) в кодировке **ANSI** текущей кодовой страницы **ОС** [8] или в кодировке **Unicode UTF-8**. При использовании **UTF-8** в начале файла нужен маркер **BOM** (byte order mark **EF**,**BB**,**BF**), согласно стандарту **RFC3629** [9]. **Алфавит** интерпретатора приведен в таблице 2.
- Строки текста разделяются символами **CR**,**LF**.

- Символ «;» точки с запятой отделяет **комментарий**, который продолжается до конца текущей строки. Комментарий при анализе строк **игнорируется** и не входит в интерпретируемый текст.
- **Незначащие пробелы** (т. е. символы пробела и табуляции) в начале и конце строки **игнорируются** (удаляются из строки).
- Входной текст разделяется на **именованные секции**. Признаком секции являются квадратные скобки **[…]**, которыми должна начинаться и заканчиваться строка (после удаления незначащих пробелов). Секция считается отдельным поименованным текстом.
- Текст секции может завершаться маркером **пустой секции** **[]**, служащей специально для разделения секций. Содержимое пустой секции при чтении игнорируется.
- **Одноименные секции объединяются** в один текст. Поэтому секция текста может быть разбита в тексте файла на несколько частей, которые при слиянии образуют единый текст секции.
- Имена файлов конфигурации не должны содержать пробелов.
- Файлы конфигурации допускают (рекурсивное) **включение** других файлов конфигурации, содержимое которых добавляется в конец, после текста текущего файла. Включение производится с помощью специальной секции с постоянным именем **[ConfigFileList]**, в которой помещается список выражений вида **ConfigFile =** *filename*, со **ссылкой** на включаемый файл.
- В **файловых ссылках** часто используются (и рекомендуются) **относительные ссылки**, т. е. имена файлов, данные относительно данного файла конфигурации, либо относительно специальных каталогов (домашний каталог пользователя **~** и пакета **~~**).
- Строки текста разделяются на **слова**. Разделителями слов служат **пробелы** « », символы **табуляции** «**Tab**» и **запятые** «**,**».
- Слова, выделяемые из строк, интерпретируются (в зависимости от контекста и дополнительных соглашений, связанных с целевым объектом) как **идентификаторы** (имена элементов данных) или как **литералы** (строковые или числовые значения элементов данных).
- Внутри секции текста элементы данных обычно идентифицируются именами, после которых стоит знак равно и литерал, содержащий значение: «**Имя = Значение**». Незначащие пробелы удаляются.
- Для **адресации** элементов данных используются: **имя файла** конфигурации, **имя секции** текста и **имя параметра** в секции, которые вместе однозначно идентифицируют любой элемент данных в конфигурации: «**Файл [Секция] Параметр**».
- **Идентификаторы** (слова, обозначающие имена структур данных) могут содержать любые печатные символы, кроме пробелов, запятой «,», комментария «;», знака равно «=» и квадратных скобок **[…]** в начале и конце слова. В именах чаще всего используются: буквы «**a**»..«**z**», цифры «**0**»..«**9**», точка «.», знак подчеркивания «_». Несколько реже используются также другие печатные символы, например: «**&**», «**@**», «**:**», «**$**», «**+**», «**-**», «***»», «**/**», «**(**», «**)**».
- Идентификаторы обычно **не чувствительны** к регистру символов, т. е. идентификаторы, заданные в верхнем и нижнем регистре обычно (если не сказано другое) считаются эквивалентными.
- Строковые литералы при чтении обычно (если не сказано другое) преобразуются к верхнему регистру символов. Это сделано для облегчения сравнения строк при анализе текста.

*Таблица 2. Алфавит интерпретатора **DaqConfig**.*

№	Символ(ы)	Область применения
1	**CR,LF**	Разделитель для строк.
2	**;**	Разделитель для комментария.
3	**Пробел « », Tab «,» (запятая)**	Разделители для слов.
4	**[…]**	Квадратные скобки для обозначения поименованной **секции** текста.
5	**=**	Декларация параметра или элемента данных. Например, **Address = 10**.
6	**0…9**	Цифры для числовых литералов и идентификаторов. Например, литералы **123**, **3.14** и идентификаторы **x1**, **y2**.
7	**$**	Признак шестнадцатеричного числа (hexadecimal), например **$FE05**.
8	**a..z, _**	Буквенные символы для идентификаторов, например: **width**, **x1**, **_y1**.
9	**. (точка)**	Разделяет составные имена или целую и дробную часть числа. Например, имя **BEAM.ENERGY** или число **3.14**.
10	**& \ / - + * ( ) . : ^ # @ % ~ !**	Дополнительные печатные символы (не буквенно-цифровые) тоже могут использоваться в идентификаторах. Например, **&DimSrv**.

*Таблица 3. Синтаксические конструкции интерпретатора **DaqConfig**.*

№	Конструкция	Описание
1	**Идентификаторы**	Имена элементов данных.
2	**Литералы**	Числовые (вещественные или целые) и строковые константы.
3	**Текст ; Комментарий**	Комментарий к тексту.
4	**[…]**	Декларация секции, т. е. поименованного фрагмента текста.
5	**Имя = Значение**	Декларация параметра или элемента данных.
6	**Список = Элемент**	Декларация элемента списка.
7	**[ConfigFileList]** **ConfigFile = *filename***	Секция включаемых файлов конфигурации. Включение (добавление) файла конфигурации в конец текста.

В таблице 2 приведен алфавит, а в таблице 3 - основные синтаксические конструкции интерпретатора **DaqConfig**. Они подробно рассматриваются в следующих разделах документа.

Исходные файлы

В языке **DaqScript** имена исходных файлов **не должны содержать пробелов**, т. к. они могут использоваться в качестве ссылок или идентификаторов. **Расширения** имен файлов должны соответствовать назначению файла. Используемые расширения перечислены в таблице 1.

Файлы конфигурации могут **включать** в себя (присоединять) другие файлы через список **ConfigFile** в секции **[ConfigFileList]**. При этом **основным** (главным) считается файл, с которого начинается загрузка конфигурации и который включает в себя все остальные файлы.

Алфавит

Текст исходных файлов читается как **поток байтовых строк**, по 8 бит на символ, разделенных символами **CR**,**LF**. Если в начале файла есть маркер **BOM** (byte order mark **EF**,**BB**,**BF**), то текст берется в кодировке **Unicode UTF-8** по стандарту **RFC3629** [9]. Иначе текст берется в кодировке **ANSI** текущей кодовой страницы **ОС** [8]. В тексте допустимы любые печатные символы. Пробелы « », табуляции «**Tab**», запятые «,», точки с запятой «;», знаки равно «=» и квадратные скобки «**[]**» имеют особый смысл, указанный в таблице 2.

Комментарии

В конфигурациях **DaqConfig** часто используются **комментарии**, которые начинаются с символа «;» точки с запятой и продолжаются до конца текущей строки текста. Комментарии игнорируются при чтении и служат для размещения пояснительной и сопроводительной информации к коду конфигурации. Например: распечатка 1.

*Распечатка 1. Пример использования комментариев **DaqConfig**.*

```
[TagList]                                    ; Секция объявления тегов:
DEMO.ZUPDC.ZUP1.POLL_ENABLE = integer 0      ; Флаг разрешения опроса
DEMO.ZUPDC.ZUP1.POLL_RATE   = real    0      ; Частота, опросов в секунду
DEMO.ZUPDC.ZUP1.ERROR_CNT   = real    0      ; Счетчик ошибок устройства
DEMO.ZUPDC.ZUP1.ID_NAME     = string  ZUP1   ; Идентификатор устройства
DEMO.ZUPDC.ZUP1.ID_REV      = string  Ver 6-33 ; Версия прошивки
DEMO.ZUPDC.ZUP1.PAR_SV      = real    0      ; Уставка напряжения, Вольт
DEMO.ZUPDC.ZUP1.PAR_AV      = real    0      ; Текущее напряжение, Вольт
[]                                           ; Конец секции тегов.
```

Комментарии, при верном использовании, улучшают ясность кода и облегчают его чтение, понимание и сопровождение. Часто комментарии содержат важную дополнительную информацию и служат частью технической документации для разрабатываемой **АСУ**.

Отделение комментария является первой операцией при анализе текста, поэтому комментировать можно любые строки текста, включая заголовки **секций**.

Секции текста […]

Входной текст файлов конфигурации логически разбивается на части — **секции**. Признаком начала секции является **заголовок секции**, которым является строка с именем секции, помещенная в **квадратные скобки**, например, **[DeviceList]**. При этом незначащие пробелы в начале и конце строки, а также комментарии игнорируются. **Имя секции не чувствительно к регистру символов** и может содержать любые печатные символы (включая пробелы), кроме точки с запятой и квадратных скобок. Важно лишь то, что **квадратные скобки должны быть крайними** (первыми и последними) **значащими** (т. е. отличными от пробелов) **символами**, после отделения комментариев.

Текст содержимого секции начинается со следующей после заголовка секции строки и продолжается до следующего заголовка секции или конца файла. Часто для завершения секции используется **пустая секция** с заголовком **[]**. Пустая секция при чтении игнорируется, поэтому её используют как **маркер конца секции**. Например: распечатка 2.

*Распечатка 2. Пример секций текста в **DaqConfig**.*

```
[DeviceList]                          ; Заголовок секции
&Demo1 = device software script       ; Содержимое секции
&Demo2 = device software program      ; Содержимое секции
[]                                    ; Маркер конца секции
[Demo - Section]                      ; Заголовок секции с именем, содержащим пробелы
Port = 1                              ; Содержимое секции
[This] is not a section               ; Ошибка, квадратные скобки не являются крайними
This is [not] a section               ; Ошибка, квадратные скобки не являются крайними
This is not a [section]               ; Ошибка, квадратные скобки не являются крайними
[]                                    ; Маркер конца секции
```

Секции рассматриваются как **логически отдельные тексты**, как бы «**файлы внутри файлов**». При адресации элементов данных имя секции участвует наряду с именем файла и именем параметра в качестве необходимых «координат» ссылки.

Объединение и разбиение одноименных секций

Одноименные секции (с одинаковым заголовком) **объединяются** в один логически единый текст. Это значит, что **содержимое секции можно разбить** в тексте файла на любое число частей, исходя из удобства чтения и понимания текста.

Например, удобно разбивать секцию **[DeviceList]** со списком устройств **DAQ** системы, располагая декларацию устройства рядом с его описанием. Например: распечатка 3.

*Распечатка 3. Пример разбиения секций текста в **DaqConfig**.*

```
[DeviceList]                          ; Секция списка устройств DAQ системы (начало)
&Demo1 = device software script       ; Декларация устройства &Demo1
[]                                    ; Маркер конца секции
[&Demo1]                              ; Секция описания устройства &Demo1
ScriptSection = [&Demo1.Script]       ; Задание имени секции с текстом сценария DaqScript
[]                                    ; Маркер конца секции
[&Demo1.Script]                       ; Секция с текстом сценария DaqScript
var RunCount                          ; Текст демонстрационного сценария:
RunCount=RunCount+1                   ; Приращение счетчика вызовов RunCount
@echo RunCount = %RunCount            ; Вывод счетчика RunCount на печать
[]                                    ; Маркер конца секции
[DeviceList]                          ; Секция списка устройств DAQ системы (продолжение)
&Demo2 = device software script       ; Декларация устройства &Demo2
[]                                    ; Маркер конца секции
[&Demo2]                              ; Секция описания устройства &Demo2
…                                     ; … и так далее
```

Включение файлов в секции [ConfigFileList]

Специальная секция **[ConfigFileList]** содержит список элементов **ConfigFile** и служит для **включения** в конфигурационный файл других конфигурационных файлов. Элементами списка **ConfigFile** являются имена включаемых файлов, т. е. **файловые ссылки**. Например: распечатка 4.

*Распечатка 4. Пример включения файлов в секции **[ConfigFileList]**.*

```
[ConfigFileList]                      ; Секция списка включаемых файлов
ConfigFile = TagList                  ; Включение файла с расширением .CFG по умолчанию
ConfigFile = Client.cfg               ; Включение файла из текущего каталога конфигурации
ConfigFile = .\Access.cfg             ; Включение файла из текущего каталога конфигурации
ConfigFile = ..\Common.cfg            ; Включение файла из родительского каталога
ConfigFile = ..\Data\Custom.ini       ; Включение файла с относительным путем
ConfigFile = ~\Settings.ini           ; Включение файла из домашнего каталога пользователя
ConfigFile = ~~\Resource\Data.ini     ; Включение файла из домашнего каталога пакета CRW-DAQ
ConfigFile = D:\Data\Params.ini       ; Включение файла с абсолютным именем файла
ConfigFile = \\storage\Params.ini     ; Включение файла с сетевым именем файла
[]
```

Тексты файлов из списка **ConfigFile** включаются **в конец текста**, после текста текущего файла. Включение файлов **рекурсивно**, т. е. каждый из включаемых файлов может содержать также другие включаемые файлы, глубина рекурсии ограничена 8 уровнями. При этом включение файлов **эксклюзивно**, т. е. каждый файл включается в текст только один раз, а запросы на повторные включения игнорируются.

Таким образом, файл конфигурации может разбиваться из соображений удобства на **основной** файл и **дополнительные** включаемые файлы, описанные в секции **[ConfigFileList]**. Чтение файлов происходит так, как будто бы файлы, из секции **[ConfigFileList]**, добавляются в конец основного файла, хотя сама секция **[ConfigFileList]** может находиться где угодно. Например: распечатка 5.

Распечатка 5. Пример разбиения файлов конфигурации на части.

```
Файл Crw32.ini:                          ; Это основной файл
[ConfigFileList]                         ; Включение файлов конфигурации:
ConfigFile = Resource\Crw32.Param.ini    ; Описание параметров   CRW-DAQ
ConfigFile = Resource\Crw32.Tools.ini    ; Описание инструментов CRW-DAQ
…                                        ; … и так далее

Файл Resource\Crw32.Param.ini:           ; Это дополнительный (включаемый) файл
[System]                                 ; Секция системных параметров
HelpFile    = Resource\Help\Crw32.hlp    ; Ссылка на файл справки
…                                        ; … и так далее

Файл Resource\Crw32.Tools.ini:           ; Это дополнительный (включаемый) файл
[@run]                                   ; Секция описания команды @run
wintail = Resource\Shell\wintail.exe     ; Расположение утилиты wintail
…                                        ; … и так далее

Пример чтения параметров:                ; Чтение с помощью API функции readini
readini('Crw32.ini [System] HelpFile')   ; Вернет строку: Resource\Help\Crw32.hlp
readini('Crw32.ini [@run] wintail')      ; Вернет строку: Resource\Shell\wintail.exe
```

Файловые ссылки

Как в секции **[ConfigFileList]**, так и в других модулях в файлах конфигураций используются **ссылки на файлы** конфигураций, мнемосхем, изображений, программ и других объектов. Типы файловых ссылок показаны в таблице 4. Прежде всего, файловые ссылки делятся на **абсолютные** и **относительные**.

Абсолютные ссылки включают **полный путь** файла, т. е. диск (или сервер для сетевого пути), путь каталога, имя файла и расширение. Абсолютные ссылки самодостаточны, т. е. не нуждаются в дополнительной информации о текущем каталоге, но они неудобны слишком сильной привязкой к структуре конкретной файловой системы, что делает их крайне неудобными на практике. Поэтому абсолютные имена используются редко и в общем случае не рекомендуются.

Относительные (неполные) файловые **ссылки** отличаются тем, что в них отсутствует часть полного пути файла (пропущен диск или сервер, а возможно, также часть пути каталога или расширение). Эти ссылки задаются **относительно рабочего каталога**, которым обычно считается каталог расположения текущего файла, в котором указана ссылка. При вычислении имен файлов по относительным ссылкам допускается использовать как общепринятые соглашения **ОС** (ссылка **.** - текущий каталог, **..** - родительский каталог), так и специальные соглашения языка **DaqConfig** (ссылка **~** - домашний каталог пользователя **%UserProfile%**, **~~** - домашний каталог работающего пакета **CRW-DAQ**, в переменной **%CRW_DAQ_SYS_HOME_DIR%**, обычно это **C:\Crw32exe**). В распечатках 6 и 7 приведены часто используемые ссылки в файлах конфигураций на локальные или общие библиотечные файлы и каталоги.

В случае отсутствующего в ссылке расширения файла ему дается **расширение по умолчанию**, которое зависит от контекста. Это должно оговариваться в каждом случае отдельно.

*Таблица 4. Типы файловых ссылок в языке **DaqConfig**.*

№	**Файловая ссылка** Пример	**Комментарий**
1	**Drive:\Path\FileName.Ext** C:\Daq32\Demo\Config\Test.cfg	**Абсолютная** файловая ссылка с заданием диска **Drive**, пути **Path**, имени файла **FileName** и его расширения **.Ext**.
2	**\\Host\Path\FileName.Ext** \\storage\Daq32\Settings.ini	**Абсолютная** сетевая ссылка с указанием сервера **Host**, пути **Path**, имени файла **FileName** и его расширения **.Ext**.
3	**FileName** demo_main_ctrl	**Относительная** файловая ссылка с заданным именем **FileName**, путем и расширением по умолчанию. Путь по умолчанию – как у текущего файла, расширение по умолчанию – зависит от контекста.
4	**FileName.Ext** demo_main_ctrl.cfg	**Относительная** файловая ссылка с заданным именем **FileName**, расширением **.Ext**, и путем по умолчанию. Путь по умолчанию – как у текущего файла.
5	**RelPath\FileName.Ext** ..\DaqPas\demo_main_ctrl.pas	**Относительная** файловая ссылка с заданным именем **FileName**, расширением **.Ext**, и **относительным** путем **RelPath**, заданным относительно текущего файла.
6	**.** .\demo_main_config.cfg	Относительная файловая ссылка на текущий каталог. Используется в относительных путях файлов. Это общепринятое соглашение **ОС**.
7	**..** ..\Data\Custom.ini	Относительная ссылка на родительский каталог. Используется в относительных путях файлов. Это общепринятое соглашение **ОС**.
8	**~** ~\UserSettings.ini	Ссылка на домашний каталог текущего пользователя. Должна стоять в начале строки. Это соглашение **DaqConfig**.
9	**~~** ~~\Temp\Custom.ini	Ссылка на домашний каталог пакета CRW-DAQ. Должна стоять в начале строки. Это соглашение **DaqConfig**.

*Распечатка 6. Часто используемые ссылки в конфигурациях ***.cfg**.*

Часто используемые в DaqConfig ссылки в файлах конфигураций *.cfg:

```
..\Bitmaps\                              ; Локальный каталог изображений        *.bmp
..\Calibr\                               ; Локальный каталог калибровок         *.cal
..\Circuits\                             ; Локальный каталог мнемосхем          *.crc
..\Config\                               ; Локальный каталог конфигураций       *.cfg
..\DaqPas\                               ; Локальный каталог программ DaqPascal *.pas
..\Data\                                 ; Локальный каталог данных *.ini *.crw *.dat
..\Help\                                 ; Локальный каталог справки      *.pdf *.htm
..\Temp\                                 ; Локальный каталог временных файлов   *.log
..\Utility\                              ; Локальный каталог утилит *.exe *.bat *.cmd
~~\Resource\DaqSite\                     ; Общий каталог библиотек файлов DAQ системы
~~\Resource\DaqSite\StdLib\Calibr\       ; Общий каталог библиотеки калибровок  *.cal
~~\Resource\DaqSite\StdLib\DaqPas\       ; Общий каталог библиотеки программ    *.pas
~~\Resource\DaqSite\StdLib\Include\      ; Общий каталог библиотеки StdLibrary  *.inc
~~\Resource\DaqSite\Default\             ; Общий каталог стандартных конфигураций
~~\Resource\DaqSite\Default\DAQ.cfg      ; Настройки по умолчанию для секции [DAQ]
~~\Resource\DaqSite\Default\Integrity.cfg ; Настройки по умолчанию для системы @Integrity
~~\Resource\DaqSite\Default\DatSrv.cfg   ; Сервер &DatSrv      с обычной настройкой
~~\Resource\DaqSite\Default\DimSrv.cfg   ; Сервер &DimSrv      с обычной настройкой
~~\Resource\DaqSite\Default\WebSrv.cfg   ; Сервер &WebSrv      с обычной настройкой
~~\Resource\DaqSite\Default\CronSrv.cfg  ; Сервер &CronSrv     с обычной настройкой
~~\Resource\DaqSite\Default\PlotSrv.cfg  ; Сервер &PlotSrv     с обычной настройкой
~~\Resource\DaqSite\Default\SpeakSrv.cfg ; Сервер &SpeakSrv    с обычной настройкой
~~\Resource\DaqSite\Default\ModbusProxy.cfg ; Сервер &ModbusProxy с обычной настройкой
~~\Resource\DaqSite\Default\DimDnsLocal.cfg    ; Настроить &DimSrv на локальный DNS
~~\Resource\DaqSite\Default\DimAccessLocal.cfg ; Разрешить &DimSrv доступ локально
~~\Resource\DaqSite\Default\DimAccessToAll.cfg ; Разрешить &DimSrv доступ для всех
```

*Распечатка 7. Часто используемые ссылки в мнемосхемах ***.crc**.*

```
Часто используемые ссылки DaqConfig для мнемосхем *.crc:

..\Bitmaps\                                ; Локальный каталог изображений        *.bmp
..\Circuits\                               ; Локальный каталог мнемосхем          *.crc
~~\Resource\DaqSite\StdLib\Bitmaps\        ; Общий каталог библиотеки изображений *.bmp
~~\Resource\DaqSite\Default\Painter.crc    ; Общая библиотека сценариев Painter   *.crc
```

На практике в прикладных конфигурациях, работающих в **DAQ** системе, чаще всего используются относительные ссылки. Это связано с тем, что относительные ссылки позволяют легко перемещать каталог **DAQ** конфигурации в любое место, не теряя его работоспособности, т. к. относительные ссылки не зависят от расположения каталога **DAQ** конфигурации. При этом необходимые для работы полные пути файлов вычисляются по относительным ссылкам с помощью функций **API**. Например, в **DaqPascal** функция **daqfileref(f,e)** вычисляет полный путь файла по относительному пути **f** и расширению по умолчанию **e**. При этом за рабочий каталог, относительно которого вычисляются ссылки, принимается каталог основного файла текущей **DAQ** конфигурации.

Литералы

Литералы служат для задания значений элементов данных в выражениях **DaqConfig**. Это постоянные значения (константы), выраженные в явной **нотации** (т. е. способе записи), а не в виде символического имени. Интерпретатор **DaqConfig** поддерживает строковые и числовые литералы, т. е. строки текста, а также целые или вещественные числа, выраженные в виде цифр. Для целых чисел допускается десятичная или шестнадцатеричная нотация (с префиксом **$**). Для вещественных используется десятичная нотация, в том числе с использованием научной нотации (с мантиссой и экспонентой после символа **E**). Строковые литералы (в зависимости от контекста) могут состоять из одного или нескольких слов. Строки из нескольких слов, могут интерпретироваться как одна строка с пробелами или как набор слов, в зависимости от контекста. Например: распечатка 8.

*Распечатка 8. Примеры литералов **DaqConfig**.*

```
[DemoSection]                              ; Начало секции
Buffer  = 1024                             ; Целое число в десятичной нотации.
Mask    = $03FF                            ; Целое число в шестнадцатеричной нотации.
Factor  = 3.1415                           ; Вещественное число в фиксированной нотации.
Scale   = 1.024E+3                         ; Вещественное число в научной нотации.
Flag1   = 1                                ; Логическое значение 1 = true.
Flag2   = 0                                ; Логическое значение 0 = false.
Command = DEMO.CMD.HELP                    ; Строковый литерал (ссылка) из одного слова.
Comment = Операция выполнена успешно.      ; Строковый литерал из нескольких слов.
[]                                         ; Конец секции
```

В выражениях **DaqConfig** также часто используются логические переменные (флаги). Обычно для них используются литералы **0** или **1**, хотя допускаются также **false** или **true**. Значения **0**/**1** используются чаще, т. к. они компактнее и привычнее.

Строковые литералы могут, в частности, содержать в себе идентификаторы элементов данных или имена секций. Это используется для **ссылки** на другие элементы данных. Например, параметр конфигурации **Command** может ссылаться на тег **DEMO.CMD.HELP**, участвующий в обработке данных.

Идентификаторы

Идентификаторы – это символьные **имена** параметров и элементов данных, под которыми они известны в системе. В отличие от многих других языков программирования, идентификаторы в **DaqConfig** имеют довольно мягкие требования. Идентификатором может быть непрерывная цепочка любых **печатных** символов, кроме символов пробела « », табуляции «**Tab**», запятой «,», точки с запятой «;», знака равно «=» и квадратных скобок **[]**. Например: распечатка 9.

*Распечатка 9. Примеры идентификаторов **DaqConfig**.*

```
[Demo]                                  ; Начало секции текста
Command1    = DEMO.CMD.HELP             ; Допустимый идентификатор с буквами и цифрами.
2Command    = DEMO.CMD.EXIT             ; Допустимый идентификатор, начинающийся с цифры.
BUFF_SIZE   = 1024                      ; Допустимый идентификатор со знаком подчеркивания.
DATA.MASK   = $03FF                     ; Допустимый идентификатор с точкой в имени.
&Device     = &DimSrv                   ; Допустимый идентификатор с символом «&».
Com-Port    = COM5                      ; Допустимый идентификатор с символом «-».
[Param]     = 3.1415                    ; Недопустимый идентификатор с квадратными скобками.
Value,2     = 1.024E+3                  ; Недопустимый идентификатор с запятой в имени.
Flag 1      = 1                         ; Недопустимый идентификатор с пробелом в имени.
[]                                      ; Конец секции текста
```

Обычно идентификаторы содержат латинские буквы и цифры, например, **Address** или **Port1**. Часто имена включают также знаки подчеркивания «_» или точку «.» для разделения длинного имени на части. Это делается для получения более ясных и удобочитаемых **составных имен**, таких как **BUFFER.SIZE** или **BUFFER_SIZE**. В именах допускаются также символы, не являющиеся буквами или цифрами, например, «**&**», «**+**», «**-**», «*****», «**:**», «**/**», «****» и другие. Эти символы можно использовать в именах, но не стоит ими злоупотреблять, чтобы не нарушать ясность и удобство чтения кода.

Проблема выбора имен при проектировании систем

Разумный выбор идентификаторов является важнейшей частью **системного проектирования** прикладного **ПО**. Это связано с тем, что имена параметров задаются на **этапе проектирования**, до написания кода или сборки оборудования. Имена первичны по отношению к другим атрибутам данных, таким как адреса, ссылки или значения, которые могут меняться от сеанса к сеансу или в процессе работы.

Ошибки в системном проектировании, такие как **конфликты имен** параметров из разных подсистем, трудно обнаруживаются и требуют больших затрат для исправления. Особенно много проблем возникает при «слиянии» двух и более подсистем в одну большую систему. В этом случае конфликтующие имена требуют трудоемкой переработки кода работающих и отлаженных систем, что нежелательно. Поэтому при выборе имен надо учитывать возможность расширения разрабатываемых систем в будущем.

Желательно, чтобы имена имели **интуитивно понятный смысл**. Это делает программный код более удобным для чтения, понимания и сопровождения. В идеале код должен быть «**самодокументированным**», не требующим дополнительных пояснений. Это достигается выбором хороших имен, делающих смысл программного кода очевидным.

Желательно, чтобы имена было **удобно произносить вслух**. Это облегчает обсуждение прикладного кода с коллегами. Непроизносимые имена затрудняют деловое общение и совместное решение задач.

Составные имена и дополнительные соглашения

По указанным выше причинам в прикладных системах часто используются **составные имена**, которые составляются из простых слов или аббревиатур, разделенных точками и знаками подчёркивания. Точки обычно используется для разделения частей имен по иерархии подсистем, а подчёркивание – для разделения слов или номеров в именах параметров, состоящих из простых слов. Так, в составном имени **PHOS.ADAM.POLL_RATE** точки разделяют уровни иерархии структуры данных (**PHOS** – имя установки, **ADAM** – подсистема, **POLL_RATE** – параметр), а знак подчеркивания (**POLL_RATE**) разделяет простые слова (poll rate, частота опроса) в имени параметра для удобства чтения.

Составные имена позволяют решить проблему **пространства имен** при проектировании систем, давая возможность разделять имена разных подсистем по **префиксу** (начальной части составного имени).

В **DAQ** системах часто используются составные имена вида **Facility.Subsystem.Device.Param**, где **Facility** – имя установки, **Subsystem** – имя подсистемы, **Device** – имя устройства, **Param** – имя параметра. Например, имя **PHOS.COOL.WATER.FLOW** может обозначать поток (flow) датчика воды (water) системы охлаждения (cool) установки (**PHOS**). В сложных системах число частей в имени может быть и больше.

В составных именах также часто используются **аббревиатуры** - сокращения, составленные из частей слов длинных названий установок или приборов. Так, в приведенном выше примере **PHOS** - аббревиатура от англ. **pho**ton **s**pectrometer (фотонный спектрометр).

Составные имена – пример **дополнительных соглашений по наименованию**, которые не являются строго обязательными, но рекомендуются для улучшения структуры кода прикладного **ПО**. Многие соглашения используются так часто, что со временем превращаются в обязательные правила.

Имена устройств DAQ системы - &DeviceName

К дополнительным соглашениям по наименованию также относятся имена устройств в **DAQ** системе, которые, по сложившейся традиции, начинаются со знака «**&**» (ampersand). Например, **&PHOS.COOL.GUI**. Все стандартные серверы **DAQ** системы (**&DimSrv**, **&WebSrv**, **&DatSrv**, **&CronSrv** и другие) следуют этому соглашению. Поскольку это соглашение поддерживается в **API**, его можно считать обязательным.

Адресация данных в DaqConfig

Три слова: 1) **имя файла**, 2) **имя секции** и 3) **идентификатор параметра**, взятые вместе, однозначно адресуют параметры и элементы данных в конфигурациях **DaqScript**. Пример: распечатка 10.

*Распечатка 10. Примеры адресации параметров в **DaqConfig**.*

```
Readini('Crw32.ini [System] HelpFile')        ; Чтение параметра HelpFile
                                              ; из секции [System] из файла Crw32,ini
Readini('..\Data\Settings.ini [COM] Port')    ; Чтение параметра Port из секции [COM]
                                              ; из файла ..\Data\Settings.ini
```

При чтении данных функций **readini** имя файла и секция могут как задаваться явно, так и определяться контекстом. Например, вызов **`readini('Address')`** означает чтение параметра (**Address**) из текущего файла конфигурации, из секции текущего устройства.

Декларация параметра

Выражения вида **Name = Value** или **Name = Value1, Value2, …** служат для декларации (объявления) параметра с **идентификатором** **Name** и присвоения ему значения, заданного **литералом** **Value** или списком литералов **Value1**, **Value2**, … - в зависимости от контекста.

Декларация параметра определяется, кроме имени **Name**, его **адресным** контекстом, т. е. **файлом** и **секцией**, в которой она присутствует. Одинаковые имена в разных секциях декларируют совершенно разные параметры.

Декларация не означает немедленное действие. Она означает, что в момент, когда параметр потребуется программе для работы, он может быть прочитан из файла и получит при чтении указанное в нем значение. Чтение может происходить в различные моменты, например, при загрузке конфигурации, нажатии кнопки или при другом событии.

*Распечатка 11. Пример декларации параметров в **DaqConfig**.*

```
[&BenchMark]                                ; Секция описания DAQ устройства &Benchmark
InquiryPeriod = 1                           ; Период опроса, миллисекунд
DevicePolling = 1, tpIdle                   ; Период и приоритет программного потока
ProgramSource = Demo_Benchmark.pas          ; Ссылка на файл с кодом программы
DebugFlags    = 3                           ; Флаги для отладки
OpenConsole   = 1                           ; Режим консоли
AnalogOutputs = 1                           ; Число аналоговых выходов
Link AnalogOutput 0 with curve bench1       ; Подключение кривой к аналоговому выходу 0
[]                                          ; Маркер конца секции
```

На распечатке 11 приведен типичный пример фрагмента файла конфигурации с декларацией параметров для **DAQ** устройства с именем **&BenchMark**. В этом примере параметр **ProgramSource** задает ссылку на файл с исходным кодом программы на языке **DaqPascal**, параметр **InquiryPeriod** задает период опроса, и т. д. Обращаем внимание, что параметры помещены в одноименную с устройством секцию **[&BenchMark]**, что является общим правилом: объекты обычно описываются в одноименной с ними секции.

Правила сканирования параметров

При чтении параметров происходит **сканирование** текста секции до **первого вхождения** декларации искомого параметра. Это значит, что если в тексте секции есть несколько деклараций одного параметра, то принята будет первая из них. Остальные декларации игнорируются. Пример показан на распечатке 12.

Распечатка 12. Пример повторяющейся декларации параметра.

```
[Demo]                                      ; Секция параметров для демонстрации
Param = Primary                             ; Декларация параметра - первое вхождение
…                                           ; Другие параметры
Param = Secondary                           ; Декларация параметра - второе вхождение
…                                           ; Другие параметры
Param = Last                                ; Декларация параметра - третье вхождение
…                                           ; Другие параметры
[]                                          ; Маркер конца секции

Readini('[Demo] Param');                    ; Возвращает значение: PRIMARY
```

Переопределение параметров и параметры по умолчанию

Правила **сканирования** параметров, вместе с правилами **объединения** секций и **включения** файлов, позволяют гибко управлять конфигурацией, задавая **параметры по умолчанию** во включаемых библиотечных файлах, сохраняя возможность их переопределения при необходимости в прикладных файлах конфигурации.

При включении файлов текст этих файлов помещается **после** текста основного файла. При чтении основного файла содержимое одноименных секций объединяется в один текст. А при сканировании секции для значения параметра берется **первое** вхождение декларации параметра. Из этих посылок следует, что декларации, размещенные в основном файле, «перекрывают» декларации, размещенные во включаемых файлах. В то же время, при отсутствии декларации параметра в основном файле, будут доступны декларации из включаемых файлов.

Это дает возможность задавать большому числу параметров значения по умолчанию и размещать их в библиотечных файлах, общих для большинства систем. При этом сохраняется возможность определять значения отдельных параметров, отличных от значений по умолчанию, в основном файле конфигурации.

Распечатка 13. Пример переопределения параметров по умолчанию.

```
Файл demo.cfg:                                      ; Файл прикладной конфигурации

[DAQ]                                               ; Секция параметров DAQ
ProcessPriorityClass = RealTime                     ; Класс приоритета процесса
[]                                                  ; Маркер конца секции

[ConfigFileList]                                    ; Включение библиотечных файлов:
ConfigFile = ~~\Resource\DaqSite\Default\DAQ.cfg    ; Параметры DAQ по умолчанию
[]                                                  ; Маркер конца секции

Файл ~~\Resource\DaqSite\Default\DAQ.cfg:           ; Файл библиотеки

[DAQ]                                               ; Параметры DAQ по умолчанию
ProcessPriorityClass = Normal                       ; Класс приоритета процесса
TimeUnits            = 3600                         ; Единицы измерения времени
…                                                   ; Другие параметры
[]

Результат чтения:

Readini('demo.cfg [DAQ] ProcessPriorityClass') ; Возвращает RealTime - переопределен
Readini('demo.cfg [DAQ] TimeUnits')            ; Возвращает 3600     - по умолчанию
```

В примере, приведенном на распечатке 13, в прикладной файл **demo.cfg** включатся библиотечный файл **Default\DAQ.cfg** со значениями по умолчанию. При чтении параметр **ProcessPriorityClass** берется из файла **demo.cfg**, где он был переопределен, а параметр **TimeUnits** берется из библиотечного файла, т. к. он не был переопределен.

Использование параметров по умолчанию, размещенных в общих файлах библиотек, позволяет значительно сократить объем прикладных конфигурационных файлов, сделав их компактнее и проще.

Декларация списка объектов

В случае, если параметры не является отдельными скалярными величинами, а образуют список из объектов определенного типа, используется **декларация списка объектов**. Эта декларация имеет две часто используемые формы – конструктора и перечисления.

Декларация в форме конструктора

В этом (наиболее распространенном) случае декларация имеет вид **Name = Type Params…**, где **Name** – имя декларируемого объекта, **Type** – идентификатор типа декларируемого объекта, **Params…** - необязательный список параметров создаваемого объекта. Декларацию этого типа можно назвать **конструктором объекта** с именем **Name** и параметрами **Params…**, потому что такие декларации часто используются для создания динамического списка сложных объектов. Имя секции (например, **[TypeList]**), в которую помещается список деклараций конструкторов объектов типа **Type**, служит идентификатором этого списка.

В случае, если создаваемый объект имеет много параметров и требует отдельного описания, это описание помещается в одноименную с объектом секцию **[Name]**. Эти правила позволяют создавать списки неограниченного объема для объектов высокой сложности.

Распечатка 14. Пример декларации списка объектов в форме конструктора.

```
[DeviceList]                          ; Секция списка описания устройств
&Demo1 = device software program      ; Конструктор устройства &Demo1 типа device
[&Demo1]                              ; Секция описания устройства &Demo1
InquiryPeriod = 1                     ; Период опроса и
…                                     ; Другие параметры
[]                                    ; Маркер конца секции
[DeviceList]                          ; Секция списка описания устройств
&Demo2 = device software script       ; Конструктор устройства &Demo2 типа device
[&Demo2]                              ; Секция описания устройства &Demo2
InquiryPeriod = 1                     ; Период опроса и
…                                     ; Другие параметры
[]                                    ; Маркер конца секции
```

В примере, приведенном на распечатке 14, объявляется список **[DeviceList]** для **DAQ** устройств (**device**), в который входят устройства **&Demo1**, **&Demo2**. Устройство с именем **&Demo1** и типом **device** создается конструктором **&Demo1 = device software program**, при этом параметры **software program** уточняют тип устройства (программа **DaqPascal**). Более детальное описание параметров устройства содержится в одноименной секции **[&Demo1]**. Аналогично описывается устройство **&Demo2**.

Декларация списка в форме конструктора объектов активно используется в **DAQ** системе: для списка устройств **[DeviceList]**, списка тегов **[TagList]**, списка кривых хранилища данных **[DataStorage]**, списка окон **[Windows]**, списка сенсоров **[SensorList]**.

Декларация в форме перечисления

В этом случае декларация имеет вид **ListId = Name** или **ListId = Name1, Name2, …**. Первым словом является идентификатор списка **ListId**, за которым, после знака **=**, следует имя **Name** объекта, включаемого в список, либо перечисление **Name1**, **Name2**, … элементов списка. Это перечисление может повторяться любое число раз, пока все объекты не будут включены в список.

Декларация в форме перечисления больше подходит для случая, когда элементы списка однородны и не требуют дополнительных параметров для описания, так что достаточно перечислить их имена.

Распечатка 15. Пример декларации списка объектов в форме перечисления.

```
[Windows]                                ; Секция описания окон
Demo.Plot = Curve_Window                 ; Конструктор окна Demo.Plot
[Demo.Plot]                              ; Описание окна Demo.plot
AxisX = Abscissa, 0, 1                   ; Надпись по оси абсциссы X
AxisY = Ordinatus, 0, 10                 ; Надпись по оси ординаты Y
CurveList = Temperature, Pressure        ; Добавление элементов в список кривых CurveList
CurveList = Amplitude, Frequency         ; Добавление элементов в список кривых CurveList
CurveList = Voltage                      ; Добавление элементов в список кривых CurveList
[]                                       ; Маркер конца секции
```

В примере, приведенном на распечатке 15, в окно **Demo.Plot** объявляется список кривых **CurveList**, заданный в форме перечисления. Это список имен кривых, отображаемых в окне. Кривые должны быть объявлены в секции описания хранилища данных **[DataStorage]**.

Декларация в форме перечисления с рекурсией

Декларация в форме перечисления может содержать рекурсию, т. е. включать в себя перечисление из другой секции. Для этого в качестве элемента перечисления указывается заголовок секции, как показано на распечатке 16.

Распечатка 16. Пример декларации списка в форме перечисления с рекурсией.

```
[Windows]                                ; Секция описания окон
Demo.Plot = Curve_Window                 ; Конструктор окна Demo.Plot
[Demo.Plot]                              ; Описание окна Demo.plot
AxisX = Abscissa, 0, 1                   ; Надпись по оси абсциссы X
AxisY = Ordinatus, 0, 10                 ; Надпись по оси ординаты Y
CurveList = Voltage, [PlotList]          ; Добавление элементов в список кривых CurveList
[]                                       ; Маркер конца секции

[PlotList]                               ; Секция со списком кривых для рисования
CurveList = Temperature, Pressure        ; Добавление элементов в список кривых CurveList
CurveList = Amplitude, Frequency         ; Добавление элементов в список кривых CurveList
[]                                       ; Маркер конца секции
```

В данном примере вместо прямого перечисления кривых в список добавляется имя секции **[PlotList]**, в которой содержится перечисление этих кривых. Перечисления с рекурсией удобны тем, что позволяют помещать списки объектов в отдельных секциях и ссылаться на них.

Прямое использование текста секций

В ряде случаев прикладной код **РПО** использует текст секций напрямую, как простой текст, без выделения идентификаторов или литералов. Например, в секции может храниться простая таблица значений параметров, разделенных пробелами или запятыми. В таких случаях прикладной код **РПО** самостоятельно выполняет анализ текста в зависимости от поставленной задачи. Средства **DaqConfig** в этом случае обеспечивают лишь чтение текста секций. Пример приведен на распечатке 19.

Программный интерфейс DaqConfig API

Кроме общих синтаксических правил, **DaqConfig** предоставляет программный интерфейс для прикладного программирования в двух видах – в виде функций **Object Pascal** и функций **DaqPascal**. Функции **Object Pascal** используются в основном в программном коде ядра пакета **CRW-DAQ**, а функции **DaqPascal** - в прикладном коде **РПО**.

DaqConfig API для Object Pascal

В задачи этого введения не входит подробное описание этого **API**, редко используемого в прикладном **ПО**, поэтому здесь приводится только краткое перечисление модулей и функций.

Функционал **DaqConfig** для **Object Pascal** содержится в основном в модулях **_STR.PAS** (строковые функции) и **_FIO.PAS** (файловые функции), расположенных в каталоге **Resource\Dcc32\SYSLIB**.

Строковые функции общего назначения **WordCount** (счетчик слов), **ExtractWord** (выделение слова по номеру), **IsSameText** (сравнение строк) служат для работы со словами при анализе текста. Функция **ScanVar** служит основной функцией сканирования буфера текста и выделения из него переменных по заданному имени и ожидаемому типу значения.

Файловые функции **ExtractTextSection**, **ExtractListSection** служат для извлечения из заданного файла текста секции с заданным именем. Функция **ExtractWordList** служит для работы с списками **конструкторов**, а **ExtractEnumWordList** – списками **перечислений**. Функции **ReadIniFileVariable**, **ReadIniFilePath** служат для чтения переменных по имени и формату.

DaqConfig API для DaqPascal

Для чтения файлов конфигураций **DaqConfig** в **DaqPascal** есть набор функций, перечисленных в распечатке 17 и обеспечивающих минимальный набор средств для решения прикладных задач.

*Распечатка 17. Список основных функций **DaqConfig API** для **DaqPascal**.*

```
Функции чтения конфигураций:
readini                 ; Чтение строки параметров из конфигурации
readinisection          ; Чтения секции текста из конфигурации
Функции анализа слов и выражений:
wordcount               ; Возвращает счетчик слов данной строки
extractword             ; Выделяет из данной строки слова
skipwords               ; Возвращает остаток строки после пропуска n слов
worddelims              ; Возвращает и задает символы – разделители для анализа слов
cookiescan              ; Выделяет из текста выражения Name = Value с заданным именем Name
Фкнкции преобразования строк и чисел:
val      и  rval        ; Преобразование строки в целое и вещественное число
ivaldef  и  rvaldef     ; Преобразование строки в число с значением по умолчанию
str                     ; Преобразование числа в строку в формате по умолчанию
strfmt                  ; Преобразование числа в строку в заданном формате
Функции работы с буферами текста:
text_new                ; Создание буфера текста
text_free               ; Удаление буфера текста
text_getln              ; Чтение строки с заданным номером из буфера текста
text_putln              ; Запись строки с заданным номером в буфер текста
text_insln              ; Вставка строки с заданным номером в буфер текста
text_addln              ; Добавление строки в конец буфера текста
text_delln              ; Удаление строки с заданным номером из буфера текста
text_numln              ; Чтение счетчика строк в буфере текста
```

Функции чтения конфигураций

function readini(arg:string):string

Функция **readini(arg)** считывает строку со значением параметра конфигурации, **адресованного** строкой аргумента **arg**. Аргумент **arg** может включать от 1 до 3 слов адреса:

- **Param** - адрес с файлом и секцией по умолчанию
- **[Section] Param** - адрес с файлом по умолчанию
- **File [Section] Param** - полный адрес параметра

Здесь **File** – имя (**файловая ссылка**) читаемого файла конфигурации, **[Section]** - имя секции, **Param** – имя параметра. Если имя файла опущено, по умолчанию берется основной файл конфигурации **DAQ** системы. Если имя секции опущено, по умолчанию берется секция **[&DevName]** по имени **&DevName** текущего (вызывающего) **DAQ** устройства. Файл и секция по умолчанию используются часто, ведь устройства обычно считывают параметры именно из своей секции.

Результатом функции является **строка** текста со значением параметра, после удаления незначащих пробелов и преобразования к верхнему регистру. Для использования параметра могут потребоваться функции выделения слов и преобразования строк в числа.

*Распечатка 18. Пример использования функции **readini**.*

```
s:=Readini('demo.cfg [DAQ] HelpFile');  // Чтение HelpFile из секции [DAQ] файла demo.cfg
ComPort:=Val(Readini('ComPort'));       // Чтение целочисленного параметра ComPort
Voltage:=rVal(Readini('Voltage'));      // Чтение вещественного параметра Voltage
```

function readinisection(txt,how:integer;cfg,sec:string):integer

Функция **readinisection(txt,how,cfg,sec)** предназначена для чтения текста секции с именем **sec** из конфигурационного файла с именем **cfg** в текстовый буфер. Параметр **txt** задает ссылку на буфер, куда будет помещен прочитанный текст, а параметр **how** – флаги (опции) чтения:

- **0** - текст передается без изменений, «как есть» (as is).
- **1** - текст преобразуется в верхний регистр.
- **2** - текст преобразуется в нижний регистр.
- **4** - удаление незначащих пробелов слева.
- **8** - удаление незначащих пробелов справа.
- **16** - удаление комментария, отделяемого точкой с запятой.

Заметим, что **readini** использует при чтении флаги **29=16+8+4+1**.

Функция **readinisection** возвращает в качестве результата значение **txt** с исходной ссылкой буфера текста, что бывает удобно для вызова конструктора текста: **txt:=readinisection(text_new,…)**.

При указании пустого имени файла **cfg** в качестве имени берется основной файл конфигурации **DAQ** системы. При указании пустого имени секции **sec** в качестве имени секции берется имя секции текущего (вызывающего) **DAQ** устройства. Это бывает удобно, т. к. чаще всего устройства считывают параметры из своей «домашней» секции.

Функция **readinisection** предполагает, что прочитанный в буфер текст будет анализироваться с помощью строковых функций и функций работы с буфером текста. Пример приведен на распечатке 19.

Функции анализа слов и выражений

Эта группа функций используется для разбиения строк на слова при анализе строк. Функция **extractword(n,s)** выделяет из строки, содержащей **wordcount(s)** слов, слово номер **n**. Словом считается непрерывная цепочка символов, отличных от символов - разделителей, узнать или задать которые можно функцией **worddelims**. Слова отделены друг от друга символами — разделителями.

function wordcount(s:string):integer

Функция **wordcount(s)** находит число слов, содержащихся в строке аргумента **s**. Если строка **s** пустая или содержит только разделители, возвращается **0**.

function extractword(n:integer;s:string):string

Функция **extractword(n,s)** - выделяет из строки **s** слово номер **n**. Значение **n** лежит в интервале **1…wordcount(s)**. Если строка **s** пустая или состоит только из разделителей, или если значение **n** не лежит в интервале **1…wordcount(s)**, возвращается пустая строка.

function skipwords(n:Integer;s:string):string

Функция **skipwords(n,s)** пропускает **n** слов в начале строки **s** и возвращает остаток строки **s**, оставшийся после пропуска **n** слов и разделителей слов.

function worddelims(d:string):string

Функция **worddelims(d)** возвращает строку, состоящую из текущего набора символов разделителей слов и задает новый набор символов разделителей слов, если заданна непустая строка **d**. Если строка **d** пустая, набор символов разделителей не меняется. Поэтому вызовом **worddelims('')** можно проверять текущий набор символов разделителей без его изменения. По умолчанию в набор символов разделителей слов входят:

- chr(32) - Space - пробел.
- chr(09) - Tab - табуляция.
- chr(13) - CR - возврат каретки.
- chr(10) - LF - перевод строки.
- chr(44) - ',' - запятая.
- chr(59) - ';' - точка с запятой.
- chr(61) - '=' - знак равенства.

function cookiescan(buff,name:string; mode:integer):string

Функция **cookiescan(buff,name,mode)** сканирует строку буфера текста **buff**, ищет в нем выражение вида **name=value**, и возвращает строку значения **value**, если такое выражение найдено. Если оно не найдено, то функция возвращает пустую строку. Параметр **mode** задает дополнительные опции сканирования. Например, **mode=ord(c)** задает дополнительный символ **c** для разделения строк (кроме **CR**,**LF**) при сканировании буфера, а **mode=256** – делает функцию чувствительной к регистру символов.

Приведенные функции разбиения строк на слова позволяют легко строить простые интерпретаторы строк. Такие интерпретаторы строк, например, могут использоваться для реализации простых протоколов связи, задания параметров, анализа содержимого файлов и т. д.

Функции преобразования строк и чисел

Эта группа функций служит для преобразования строк текста в числа и наоборот. Хотя эту группу можно отнести к функциям общего назначения, эти функции так важны, что их уместно здесь привести.

function val(s:string):integer

Функция **val(s)** преобразуют строку **s** в целое число или возвращает **0**, если преобразование невозможно.

function rval(s:string):real

Функция **rval(s)** преобразуют строку **s** в вещественное число или возвращает **NaN**, если преобразование невозможно.

function ivaldef(s:string; def:integer):integer

Функция **val(s)** преобразуют строку **s** в целое число или возвращает значение **def**, если преобразование невозможно. Эта функция удобна, когда идет чтение параметра со значением, заданным по умолчанию.

function rvaldef(s:string; def:real):real

Функция **rval(s)** преобразуют строку **s** в вещественное число или возвращает значение **def**, если преобразование невозможно. Эта функция удобна, когда идет чтение параметра со значением, заданным по умолчанию.

function str(v:real):string

function str(v:integer):string

Функция **str(v)** преобразуют число **v** в строку в формате по умолчанию - **%d** для целых и **%g** для вещественных.

function strfmt(fmt:string; v:real):string

function strfmt(fmt:string; v:integer):string

function strfmt(fmt:string; v:string):string

Функция **strfmt(fmt,v)** преобразуют число или строку **v** в строку в формате, заданном строкой **fmt**. В строке формата **fmt** используются правила, аналогичные функции **Format** в языке **Object Pascal** или **sprintf** в языке **C**.

Функции работы с буферами текста

Эти функции позволяют работать с текстом в буфере памяти как с массивом строк, индексируемым номером строки. Буфер текста задается целочисленной ссылкой, создаваемой конструктором **text_new** и освобождаемой деструктором **text_free**. Строки нумеруются с **0**.

function text_new:integer

Функция – конструктор **text_new** создает объект «буфер текста» и возвращает его ссылку, используемую всеми остальными функциями работы с текстом.

function text_free(ref:integer):boolean

Функция – деструктор **text_free(ref)** освобождает буфер текста со ссылкой **ref**, созданный конструктором **text_new**.

function text_numln(ref:integer):integer

Функция **text_numln(ref)** возвращает число строк текста в буфере со ссылкой **ref**.

function text_getln(ref:integer;i:integer):string

Функция **text_getln(ref,i)** читает строку с индексом **i** из буфера со ссылкой **ref**. Строки индексируются с **0**.

function text_putln(ref:integer;i:integer;s:string):boolean

Функция **text_putln(ref,i,s)** записывает стоку **s** по индексу **i** в буфер со ссылкой **ref**. Строки индексируются с **0**.

function text_insln(ref:integer;i:integer;s:string):boolean

Функция **text_insln(ref,i,s)** вставляет строку **s** по индексу **i** в буфер со ссылкой **ref**. Строки индексируются с **0**.

function text_addln(ref:integer;s:string):boolean

Функция **text_add(ref,s)** добавляет строку **s** в конец буфера со ссылкой **ref**.

function text_delln(ref:integer;i:integer):boolean

Функция **text_delln(ref,i)** удаляет строку с индексом **i** из буфера со ссылкой **ref**.

На распечатке 19 показан простой пример использования функций **DaqConfig API**, в котором в буфер памяти читается текст заданной секции и интерпретируется как таблица значений (x,y) с двумя столбцами. Таблица считывается и выводится на печать.

*Распечатка 19. Пример использования функций **DaqConfig API**.*

```
//
// Процедура читает и печатает таблицу (x,y) из заданной секции конфигурации.
//
Procedure ReadTable(FileName,Section:String);
var ref,i:Integer; s:String; x,y:Real; b:boolean;
begin                                                  // Создаем буфер текста
 ref:=readinisection(text_new,29,FileName,Section);    // и читаем секцию в буфер
 for i:=0 to text_numln(ref)-1 do begin                // В цикле по строкам:
  s:=text_getln(ref,i);                                // Читаем строку из буфера
  x:=rVal(ExtractWord(1,s));                           // Выделяем и преобразуем x
  y:=rVal(ExtractWord(2,s));                           // Выделяем и преобразуем y
  writeln(strfmt('x=%7g',x),strfmt(' y=%7g',y));       // Используем (печатаем) x,y
 end;
 b:=text_free(ref);                                    // В конце освобождаем буфер
end;

ReadTable('demo.cfg','[DemoSection]');                 // Пример вызова процедуры
```

Приведенных сведений и примеров вполне достаточно для первого знакомства с **DaqConfig API**. Подробное описание функций **API** имеется в **online HTML** справке пакета, и повторять его нет необходимости.

Конфигурация DAQ системы

Одним из основных применений языка **DaqConfig** является описание конфигурации **DAQ** системы. Здесь приводятся краткие сведения о структуре файлов конфигурации **DAQ** системы.

Секция [DAQ]

Секция **[DAQ]** содержит параметры **DAQ** системы общего назначения, например, единицы измерения времени **TimeUnit**, или расположение файла справки **HelpFile**. Эта секция обязательно присутствует в любой конфигурации **DAQ** системы.

Обычно большая часть параметров общего назначения в разных системах совпадает. Поэтому настоятельно рекомендуется включать в любую конфигурацию файлы параметров «по умолчанию» (**Default\DAQ.cfg**, **Default\Integrity.cfg**), как показано на распечатке 20:

Распечатка 20. Пример включения конфигурации по умолчанию.

```
[ConfigFileList] ; Default DAQ settings
ConfigFile = ~~\Resource\DaqSite\Default\DAQ.cfg
ConfigFile = ~~\Resource\DaqSite\Default\Integrity.cfg
[]
```

Включение этих файлов, содержащих разумные значения параметров общего назначения, подходящих для большинства систем, позволит значительно сократить число описаний параметров в секции **[DAQ]**, т. к. большинство параметров могут быть взяты из этих файлов «по умолчанию».

Секция [TagList]

Секция **[TagList]** содержит **список тегов**, т. е. общих скалярных переменных для хранения чисел или строк. В отличие от кривых, теги не буферизуются при записи, а меняются немедленно. Теги применяются, когда играет роль текущее значение данных, а не их история изменений. Пример приведен на распечатке 21.

*Распечатка 21. Конструктор тегов в секции **[TagList]**.*

```
[TagList]                           ; Секция «список тегов»
Identifier = Type Value             ; Конструктор тега типа Type = {integer, real, string}

    где:

    Identifier  - Имя (идентификатор) декларируемого тега
    Type        - Тип тега (конструктор) из списка {integer, real, string}
    Value       - Литерал, содержащий начальное значение тега указанного типа

Например:

[TagList]                           ; Секция описания тегов
DEMO.PORT = integer 2               ; Тег типа integer (целый) с десятичным значением
DEMO.ADDR = integer $300            ; Тег типа integer (целый) с 16-ричным значением
ADC.GAIN  = real    2.5             ; Тег типа real   (вещественный)
ADC.NAME  = string  PCL-818L        ; Тег типа string (строковый)
[]                                  ; Маркер конца секции
```

Секция [DataStorage]

В секции **[DataStorage]** (от **data storage** - хранилище данных) декларируются динамические массивы данных – то есть **кривые**, по терминологии пакета **CRW-DAQ**. Кривые (**curve**) – это динамические буферы для накопления и хранения данных в виде массива точек **(x,y)**. Обычно по оси абсцисс **x** берется время, а по оси ординат **y** – измеряемая физическая величина. Кривые применяются, если в решаемой задаче интерес имеют не только мгновенные значения переменных, но и история их изменений во времени.

Секция **[DataStorage]** содержит список в форме **конструктора Curve** для декларации кривых, как показано на распечатке 22.

*Распечатка 22. Конструктор кривых в секции **[DataStorage]**.*

```
[DataStorage]                                          ; Секция «хранилище данных»
Identifier = Curve Length Step Color Marker Line       ; Конструктор для кривых Curve

    где:

    Identifier  - Имя (идентификатор) декларируемой кривой
    Curve       - Зарезервированное слово (конструктор)
    Length      - Начальная длина кривой, точек
    Step        - Шаг приращения длины кривой при добавлении данных, точек
    Color       - Цвет кривой, 0…15 или Black…White, см. рисунок 2
    Marker      - Тип маркера, 0…15, см. рисунок 2
    Line        - Тип линии, 0…15, см. рисунок 2
Цвет Color кривой может принимать значения:
0=Black, 1=Blue, 2=Green, 3=Cyan, 4=Red, 5=Magenta, 6=Brown, 7=LightGray,
8=DarkGray, 8=LightBlue, 10=LightGreen, 11=LightCyan, 12=LightRed, 13=LightMagenta, 14=Yellow, 15=White.

Например:

[DataStorage]
ao7018_1_0 = Curve  0 1024  LightRed   15 1 ; Канал 0
ao7018_1_1 = Curve  0 1024  LightBlue  15 1 ; Канал 1
ao7018_1_2 = Curve  0 1024  LightGreen 15 1 ; Канал 2
ao7018_1_3 = Curve  0 1024  Magenta    15 1 ; Канал 3
ao7018_1_4 = Curve  0 1024  Cyan       15 1 ; Канал 4
ao7018_1_5 = Curve  0 1024  Yellow     15 1 ; Канал 5
ao7018_1_6 = Curve  0 1024  White      15 1 ; Канал 6
ao7018_1_7 = Curve  0 1024  Grown      15 1 ; Канал 7
ao7018_1_8 = Curve  0 1024  Black      15 1 ; Температура холодного спая
[]                                            ; Маркер конца секции
```

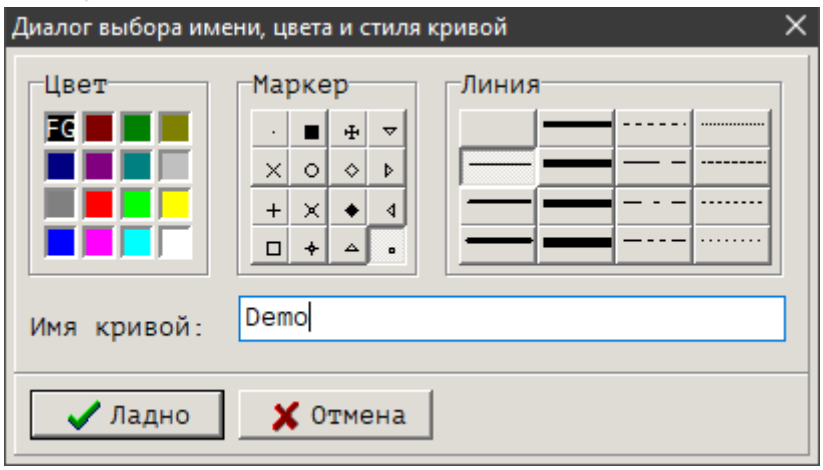

Рисунок 2. Диалог выбора стиля кривой.

Секция [DeviceList]

Секция [Windows]

Заключение

Интерпретатор **DaqConfig** является важнейшим компонентом пакета **CRW-DAQ**, выполняющим целый ряд критически значимых функций.

Он обеспечивает возможность описания конфигурации пакета **CRW-DAQ**, определяя параметры его запуска и режимов работы.

Он обеспечивает возможность …

Практика показала высокую надежность и эффективность работы интерпретатора **DaqConfig**, а также его гибкость и удобство в решении практических задач.

Ввиду всего перечисленного, освоение языка **DaqConfig** является необходимой частью подготовки разработчика прикладного **ПО** в пакете **CRW-DAQ**.

Остается пожелать читателям успеха в освоении и использовании **DaqConfig**.

Спасибо за внимание.

Список сокращений

DAQ	Data AcQuisition – сбор данных. Общепринятый термин.
CRW	CuRves in Windows – окна с кривыми. Форма представления данных.
CRW-DAQ	CuRves in Windows for Data AcQuisition – название пакета [1].
DAQ система	Система разработки и исполнения для задач сбора данных и управления.
ОС, OS	Операционная система. Operation System. Общепринятый термин.
СКУ, АСУ, АСКУ	Автоматизированная Система Контроля и Управления.
ПО	Программное обеспечение. Общепринятый термин.
РПО	Рабочее программное обеспечение - **ПО** для конкретной прикладной **СКУ**.
px	Pixel. Пиксель. Единица измерения координат растровых изображений.
pt	Point. Пункт. Единица измерения размера шрифтов.
FIFO	First In, First Out. Очередь типа «первым вошел — первым вышел».
API	Application Program Interface. Прикладной программный интерфейс.
GUI	Graphical User Interface. Графический интерфейс пользователя.
DaqConfig	Интерпретатор языка описания конфигураций в пакете CRW-DAQ.
DaqScript	Интерпретатора командного языка в пакете CRW-DAQ.
DaqPascal	Компилятор языка прикладного программирования в пакете CRW-DAQ.
online	«На линии» - режим обработки событий при подключении каналов связи.
runtime	«Исполнительная система» - код программы во время выполнения задач.
realtime	«Реальное время» - задачи, критично зависящие от времени отклика **ПО**.
интерактивный	Режим работы программы при взаимодействии с пользователем **online**.
NaN	«Not a Number». Значение «не число» для обозначения ошибок.
INF	«Infinity». Значение «бесконечность» для обозначения ошибок.

Список источников

1. А.В.Курякин, Ю.И.Виноградов. Программа для автоматизации физических измерений и экспериментальных установок (CRW-DAQ). // Свидетельство РФ об официальной регистрации программы для ЭВМ № 2006612848 от 10.08.2006 г.
   Домашний сайт пакета CRW-DAQ: http://crw-daq.su
2. А.В.Курякин. Автоматизация физических экспериментов на тритиевых комплексах исследовательских установок «ТРИТОН», «АКУЛИНА» и «ПРОМЕТЕЙ». Диссертация на соискание степени кандидата физико-математических наук, Саров, 2010.
   Диссертация: http://crw-daq.su/download/documents/alexey-kuryakin-disser-final.pdf
   Автореферат: http://ftp.jinr.ru/dissertation/Kuryakin.pdf
3. А.В.Курякин, Ю.И.Виноградов. Программное обеспечение автоматизированных измерительных систем в области тритиевых технологий. // ВАНТ, серия «Термоядерный синтез», 2008 г., выпуск 2, с.80-90.
4. Протокол DCON. https://www.bookasutp.ru/Chapter2_10.aspx
   https://www.icpdas.com/download/7000/whatisdconprotocol_eng.htm
5. FP-QUI notification tool for Windows.
   https://sourceforge.net/projects/fp-qui/
   https://github.com/bfour/FP-QUI
   http://fp-qui.sourceforge.net/
6. Система «всплывающих» уведомлений FPQUI.
   https://sourceforge.net/projects/fpqui/
7. Clara Gaspar. DIM - Distributed Information Management System.
   https://dim.web.cern.ch/
8. Windows Code Pages.
   https://docs.microsoft.com/en-us/windows/win32/intl/code-pages
   Windows Code Page Identifiers.
   https://docs.microsoft.com/en-us/windows/win32/intl/code-page-identifiers
9. UTF-8, a transformation format of ISO 10646.
   https://tools.ietf.org/html/rfc3629
10. Н.Вирт. «АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ». М.: Мир, 1989, 360 стр.
    http://tka4.org