CODESYS String Libraries Package

*Данная статья является переводом одноименного фрагмента онлайн-справки CODESYS. В рамках статьи рассматривается пакет **CODESYS String Libraries**, содержащий дополнительные библиотеки для работы со строками (в частности, для работы со строками в кодировке UTF-8). Минимальная версия CODESYS, в которую может быть установлен пакет – **3.5.18.0**. Версия пакета на момент перевода – **4.0.0.0**. Его можно загрузить по ссылке.*

Вступление

Библиотеки из пакета **CODESYS String Libraries** можно использовать для обработки строк в кодировке UTF-8. Ключевым элементом является интерфейс IString из библиотеки String Segments. Каждая обрабатываемая строка должна быть представлена в программе как экземпляр этого интерфейса. Далее эти экземпляры передаются в качестве входных аргументов функциям библиотек. Для создания экземпляра интерфейса IString используется функциональный блок UTF8String из библиотеки Generic String Base.

В состав пакета входят следующие библиотеки:

- UTF-8 Encoding Support – базовые функции для обработки символов («рун») в кодировке UTF-8;
- UTF-16 Encoding Support – базовые функции для обработки символов («рун») в кодировке UTF-16;
- String Builder – эффективная обработка строк в кодировке UTF-8 (в частности, объединение строк);
- String Builder Base – базовый набор объектов, использованных в реализации библиотеки **String Builder**;
- String Segments – базовые функции и функциональные блоки для обработки экземпляров интерфейса IString;
- String Conversion – функции преобразования кодировок;
- String Functions – набор функций для обработки экземпляров интерфейса IString, соответствующий набору функций из библиотеки Standard (которые работают с типом **STRING**);
- Unicode Support – функции для определения категории символа Unicode;
- Unicode Data и Unicode Utilities – набор объектов, использованных в реализации библиотеки **Unicode Support**;
- Generic String Base – функции для обработки строк в кодировке UTF-8, объявленных как константы-дженерики (в области VAR_GENERIC CONSTANT).

Преимущества использования новых строковых библиотек

Новые библиотеки, помимо поддержки кодировки UTF-8, могут эффективно обрабатывать длинные строки – поэтому они подходят для обработки текстовых файлов, строк HTTP-запросов и т. п.

Другие преимущества связаны с поддержкой кодировки UTF-8:

- **UTF-8** поддерживает весь набор символов таблицы Unicode;
- **UTF-8** широко используется в Интернете и рекомендуется к применению World Wide Web Consortium (W3C);
- **UTF-8** совместим с кодировкой с ASCII;
- **UTF-8** имеет высокую степень функциональной совместимости (то есть его реализация на различных платформах одинакова, что позволяет передавать текст в этой кодировке между устройствами, реализованными на базе разных платформ, без каких-либо потерь);
- **UTF-8** позволяет эффективно использовать память за счет компактного кодирования символов с использованием переменного размера символа (от 1 до 4 байт).

Новые библиотеки позволяют работать со строками как с «объектами» – по принципу, который привычен для языков программирования высокого уровня:

```
// определение длины строки с помощью метода

udiStringLen := myString.Len();
IF udiStringLen = 22 THEN
...
```

ПРИМЕЧАНИЕ
Новые библиотеки не являются полной заменой «стандартных» библиотек **Standard** и **Standard64**. Тем не менее, в новых (создаваемых с нуля) проектах рекомендуется использовать именно новые библиотеки.

Начиная с версии **CODESYS V3.5 SP18 (3.5.18.0)** компилятор может интерпретировать переменные типа **STRING** как строки в кодировке UTF-8. Для этого нужно в меню **Проект** – **Установки проекта** – **Опции компиляции** установить галочку **UTF8 Encoding for STRING**. Если интерпретация требуется только для конкретных переменных (но не для всех переменных проекта), то нужно при объявлении этих переменных использовать специальный атрибут:

```
{attribute 'monitoring_encoding' := 'UTF-8'}
sValue : STRING(140) :=
   UTF8#'Đα ŝтєħ íćℌ ηuη, i¢ℌ αямєґ Tør‼ Ụn̆d ƀín̆ $☺ ќℓυℊ αł$ ωïe ƺυ√©г';
```

Благодаря возможностям кодировки UTF-8 вам больше не обязательно использовать тип данных **WSTRING** в CODESYS для работы с символами Unicode. Кодировка UCS-2, на которой основан тип **WSTRING**, в сопоставимых случаях потребует больше памяти, чем кодировка UTF-8. В кодировке UCS-2 один символ ВСЕГДА занимает 2 байта (т. е. один **WORD**) и может представлять собой только символ Unicode из диапазона от U+0000 до U+D800 и от U+DFFF до U+FFFD. В кодировке UTF-8 один символ занимает от одного до четырех байтов и может представлять собой ЛЮБОЙ символ Unicode.

Из-за того, что в кодировке UTF-8 разные символы кодируются разным числом байт, попытка обращения к строке в кодировке UTF-8 по индексу для доступа к конкретному символу не приведет к ожидаемому результату:

```
byValue := sValue[13]; // Для строки в кодировке UTF-8 вы не сможете таким образом...
                       // ... обратиться к 13-му символу строки
xOk := byValue <> 16#75; // Проверка, является ли данный символ символом 'u'
```

В этом случае для обращения к конкретному символу строки вам нужно сначала итеративно обработать ее:

```
VAR
    udiIndex, udiLength : UDINT;
    diRune : UTF8.RUNE;
    xOk : BOOL;
END_VAR

WHILE (diRune := TO_DINT(sValue[udiIndex])) <> 0 DO
    IF diRune > 16#7F THEN
        diRune := UTF8.DecodeRune(ADR(sValue[udiIndex]), 4, udiLength=>udiLength);
    ELSE
        // Мы знаем, что ASCII-символы в кодировке UTF-8 кодируются одним байтом
        udiLength := 1;
    END_IF
    IF diRune = 16#75 THEN
        EXIT;
    END_IF
    udiIndex := udiIndex + udiLength;
END_WHILE

xOk := sValue[udiIndex] = 16#75;
```

Недостатки «стандартных» библиотек для работы со строками

В строковых функциях из библиотек **Standard** и **Standard64** передача данных происходит «по значению» – то есть переменные, переданные на входы функции при ее вызове, копируются во входные переменные функции, а потом результат работы функции копируется на ее выход.

```
VAR
    sValue : STRING;
END_VAR

sValue := CONCAT(CONCAT(CONCAT('Da steh ich nun,', ' ich armer Tor!'), ' Und bin so'),
   ' klug als wie zu vor');
//                                -> Copy, LEN       -> Copy, LEN        -> Copy, LEN
//  -> Copy, LEN
//        -> 2xCopy, LEN
//               -> 2xCopy, LEN
//                     -> 2xCopy, LEN
```

Перед обработкой строк в данных функциях происходит определение их длины путем поиска в строке терминирующего символа с ASCII-кодом 16#00. Эти вычисления, а также многочисленные операции копирования, делают эти функции ресурсоемкими – и именно поэтому максимальная длина строк, которые способны обрабатывать такие функции, ограничена **255** символов.

Использование интерфейса IString

Библиотека String Segments включает в себя интерфейс IString, который хранит информацию о конкретной строке и предоставляет доступ к строке и информации о ней по ссылке. В этом заключается принципиальное отличие новых строковых библиотек от библиотек **Standard** и **Standard64**.

Следует отметить, что размер в байтах строки в кодировке UTF-8 не может быть меньше 4 и больше 16#FFFFFF (это соответствует диапазону **положительных** значений типа **UDINT**).

Интерфейс IString предоставляет следующую информацию о строке (эта информация обновляется автоматически при изменении содержимого строки):

- указатель на сегмент памяти, выделенный под размещение строки (см. метод GetSegment);
- размер этого выделенного сегмента памяти в байтах (см. выход **udiSize** метода GetSegment);
- число байт, которое используется для хранения **текущего** содержимого строки (см. метод Len). Может быть меньше размера сегмента памяти;
- признак корректности кодов Unicode всех символов строки (см. метод IsValid);
- число символов строки (см. функцию RuneCount);

```
VAR
    itfString : STR.IString;
    udiLength, udiSize, udiRuneCount : UDINT;
    pbySegment : POINTER TO BYTE;
    xValid : BOOL;
END_VAR

// Размер текущего содержимого строки в байтах
udiLength := itfString.Len();

// Адрес сегмента памяти, выделенного под хранение строки, и размер сегмента в байтах
pbySegment := itfString.GetSegment(udiSize=>udiSize);

// Число символов строки
udiRuneCount := STR.RuneCount(itfString);

// Признак того, что все коды символов данной строки являются корректными
xValid := itfString.IsValid();
```

СВЯЗЬ МЕЖДУ ТЕРМИНАМИ «СИМВОЛ» И «РУНА»

Термин «руна», используемый в документации библиотек и названиях их объектов, в целом соответствует понятию «кодовая точка Unicode» с одним интересным дополнением.

В библиотеках «руна» представляет собой псевдоним (**ALIAS**) для типа **DINT**. В результате разработчик может явно видеть, какое целочисленное значение соответствует той или иной кодовой точке. Данное значение называется «рунической константой».

Пример: символу WSTRING#"⌘" соответствует руна со значением DINT#16#2318.

Информация для экспертов

ПРИМЕЧАНИЕ
В версиях CODESYS младше **V3.5 SP18 (3.5.18.0)** отсутствует поддержка кодировки UTF-8 для типа **STRING** и, соответственно, нет возможности использовать библиотеку Generic String Base. Тем не менее, вы можете использовать другие библиотеки из пакета **CODESYS String Libraries**, но работать со строками в кодировке UTF-8 придется с помощью массивов байт:

```
VAR
    myString : STR._UTF8String;

    // Этот массив байт соответствует UTF-8 строке "CØD€šŸŠ ïß värÿ çôõl"
    abyValue : ARRAY[0..32] OF BYTE := [ // UTF-8 CØD€šŸŠ ïß värÿ çôõl
        16#43, 16#C3, 16#98, 16#44, 16#E2, 16#82, 16#AC, 16#C5, 16#A1, 16#C5, 16#B8,
        16#C5, 16#A0, 16#20, 16#C3, 16#AF, 16#C3, 16#9F, 16#20, 16#76, 16#C3, 16#A4,
        16#72, 16#C3, 16#BF, 16#20, 16#C3, 16#A7, 16#C3, 16#B4, 16#C3, 16#B5, 16#6C
    ];
    itfString : STR.IString := STR.SetSegment(myString, ADR(abyValue),
       SIZEOF(abyValue), SIZEOF(abyValue));
END_VAR
```

Преобразование STRING в IString

Структура данных для управления свойствами сегмента строки реализована с помощью функционального блока _UTF8String из библиотеки String Segments. Следует выделить память для экземпляра этого функционального блока и для самого сегмента строки.

```
VAR CONSTANT
    c_udiLength : UDINT := MAX(4, 512); // Размер сегмента памяти в байтах
    c_udiXWORD : UDINT := SIZEOF(__XWORD);
    c_udiMaxIndex : UDINT := (SIZEOF(STR._UTF8String) + c_udiLength +
       c_udiXWORD  - 1) / c_udiXWORD;
END_VAR

VAR
    axwMemory : ARRAY[0..c_udiMaxIndex] OF __XWORD;
    udiSize : UDINT;
    itfString : STR.IString := STR.CreateString(
        ADR(axwMemory), SIZEOF(axwMemory),
        ADR(UTF8#'Ða šтeђ ĭćℌ ηuη, i¢ℌ αямeґ Tørґ‼ Ụn̈d bín $☺ ḱℓuℊ αł$ ωïe ƺυ√©ґ'),
        udiStrSize=>udiSize
    );
    xOk : BOOL;
END_VAR

xOk := (
    udiSize >= 512 AND
    itfString.Len() = 133 AND
    STR.RuneCount(itfString) = 61 AND
    itfString.IsValid()
);
```

В функциональных блоках библиотеки Generic String Base инкапсулированы многие подобные низкоуровневые операции (в частности, правильное выделение и выравнивание памяти), что упрощает жизнь разработчику.

Сценарии использования

Библиотека Generic String Base предназначена для максимального упрощения обработки строк. Но при этом она использует область VAR_GENERIC CONSTANT, которая поддерживается только в версии **CODESYS V3.5 SP18 (3.5.18.0)** и выше.

Библиотека Generic String Base

В примере ниже объявляется экземпляр (**myString**) ФБ UTF8String из библиотеки Generic String Base с размером сегмента памяти 128 байт и инициализируется значением типа **STRING** (значение «1968» в виде римских цифр). Также демонстрируется использование функции RuneCount из библиотеки String Segments.

```
VAR
                                        // «1968» римскими цифрами
    myString : GSB.UTF8String<128> := (sValue := UTF8#'ⅯⅭⅯⅬⅩⅧ');
    pbySegment : POINTER TO BYTE;
    udiSize : UDINT;
    xOk : BOOL;
END_VAR

pbySegment := myString.GetSegment(udiSize=>udiSize);

xOk := (
    myString.IsValid() AND // Все символы строки являются валидными для UTF-8
    udiSize = 128 AND // Размер сегмента памяти
    myString.Len() = 17 AND // Размер текущего значения строки в байтах
    STR.RuneCount(myString) = 6 // Длина текущего значения строки в символах
);
```

Пример использования методов ФБ Builder из библиотеки Generic String Base. В примере создается экземпляр ФБ с начальным размером сегмента памяти 64 байта (**udiInitialCapacity**), который может быть динамически увеличен на 50% (**usiExtensionFactor**) в том случае, если для хранения значения строки потребуется больше памяти. В качестве строки используется то же значение, что и в предыдущем примере (UTF8#'𝕄ⅭⅯℒ✖Ⅷ').

```
VAR
                                      // «1968» римскими цифрами
    myString : GSB.UTF8String<20> := (sValue := UTF8#'𝕄ⅭⅯℒ✖Ⅷ');
    sValue : STRING := 'wurden in Mexico-Stadt die';
    wsValue : WSTRING := "ⅩⅨ.";
    diSpace : STR.RUNE := 32;
    myValue : GSB.UTF8String<128> := (sValue := UTF8#'Ọℓƴμρ☤š̊ç̍нεη $$ρïe̥łℯ
       αƅℊε𝕳αʟⅼeῆ.');

    myBuilder : GSB.Builder<(*udiInitialCapacity*) 64,
      (*usiExtensionFactor*) 50> := (itfString:=myString);
    myResult : GSB.UTF8String<128>;

    {attribute 'monitoring_encoding' := 'UTF-8'}
    sResult : STRING(128) := UTF8#'𝕄ⅭⅯℒ✖Ⅷ wurden in Mexico-Stadt die ⅩⅨ.
       Ọℓƴμρ☤š̊ç̍нεη $$ρïe̥łℯ αƅℊε𝕳αʟⅼeῆ.';

    pbyValue : POINTER TO BYTE;
    psResult : POINTER TO STRING;
    udiLength : UDINT;
    xOk : BOOL;
END_VAR

myBuilder.WriteRune(diSpace);
myBuilder.WriteString(sValue);
myBuilder.WriteRune(diSpace);
myBuilder.WriteWString(wsValue);
myBuilder.WriteRune(diSpace);
myBuilder.WriteIString(myValue);

udiLength := myBuilder.Len(); // Число байт памяти, используемых экземпляром ФБ

// Отдельные фрагменты строки, записанные выше, склеиваются и копируются в myResult
myBuilder.ToIString(myResult);

psResult := pbyValue := myResult.GetSegment();

// Запись терминирующего нуля для STRING
pbyValue[udiLength] := 0;

// Содержимое обоих областей памяти должно совпадать
xOk := (psResult^ = sResult);
```

Обработка считанного файла с помощью ФБ Builder из библиотеки Generic String Base:

```
VAR
    sPath : STRING := 'myFilePath';
    hFile : RTS_IEC_HANDLE := RTS_INVALID_HANDLE;
    myBuilder : GSB.Builder<(*udiInitialCapacity*) 16#10000,
      (*usiExtensionFactor*) 50>;
    abyBuffer : ARRAY[0..4095] OF BYTE;
    pbyData : POINTER TO BYTE;
    udiSize : UDINT;
    udiCount : UDINT;
    eEncoding : SCV.ENCODING;
    eErrorID : SCV.ERROR;
    udiResult : RTS_IEC_RESULT;
END_VAR

hFile := SysFileOpen(sPath, ACCESS_MODE.AM_READ, ADR(udiResult));
IF udiResult <> ERRORS.ERR_OK THEN
    // Обработка ошибки открытия файла
    RETURN;
END_IF

REPEAT
    pbyData := ADR(abyBuffer);
    udiSize := TO_UDINT(SysFileRead(hFile, pbyData, XSIZEOF(abyBuffer),
ADR(udiResult)));
    IF udiResult <> ERRORS.ERR_OK THEN
        // Обработка ошибки чтения из файла
        EXIT;
    END_IF

    // Определение кодировки файла
    udiCount := SCV.DecodeBOM(pbyData, udiSize, eEncoding=>eEncoding,
      eErrorID=>eErrorID);
    IF eErrorID <> 0 THEN
        // Обработка ошибки определения кодировки
        EXIT;
    END_IF

    pbyData := pbyData + udiCount;
    udiSize := udiSize - udiCount;

    WHILE udiSize > 0 DO
        // Конвертация содержимого файла в UTF-8 и его передача в myBuilder
        udiCount := myBuilder.WriteMemSegment(pbyData, udiSize, eEncoding,
           eErrorID=>eErrorID);
        IF eErrorID <> 0 THEN
            // Обработка ошибок
            EXIT;
        END_IF

        pbyData := ADR(abyBuffer);
        udiSize := TO_UDINT(SysFileRead(hFile, pbyData, XSIZEOF(abyBuffer),
           ADR(udiResult)));
        IF udiResult <> ERRORS.ERR_OK THEN
            // Обработка ошибок
            EXIT;
        END_IF
    END_WHILE
UNTIL TRUE
END_REPEAT

IF hFile <> RTS_INVALID_HANDLE THEN
    SysFileClose(hFile);
    hFile := RTS_INVALID_HANDLE;
    udiCount : UDINT;
END_IF
```

Анализ содержимого строки:

```
VAR
    myRange : SBD.Range := (itfBuilder := myBuilder);
    diRune : SBD.RUNE;
    eError : STR.ERROR;
END_VAR

myRange.Reset();
WHILE (diRune := myRange.GetNextRune(eErrorID=>eErrorID)) <> 0 AND eErrorID = 0 DO
    IF UC.IsSpace(diRune) THEN
        // Подсчет символов, которые в рамках Unicode считаются пробелами
        udiCount := udiCount + 1;
    END_IF
END_WHILE
```

Для передачи содержимого строки в кодировке UTF-8 промежуточные преобразования кодировки не требуются, поскольку данные уже закодированы в UTF-8 в экземпляр ФБ Builder. Таким образом, содержимое буфера экземпляра блока может быть скопировано куда-то напрямую, например, через TCP/IP-соединение:

```
VAR
    itfConnection : NBS.IConnection;
    pbySegment : POINTER TO BYTE;
    udiSize : UDINT;
    eError : NBS.ERROR;
END_VAR

(* Где-то ранее было установлено TCP-соединение и инициализирован itfConnection *)

pbySegment:= myBuilder.GetFirstSegment(udiSize=>udiSize, eErrorID=>eErrorID);

WHILE pbySegment <> 0 AND eErrorID = 0 DO
    eError := itfConnection.Write(pbySegment, udiSize, udiCount=>udiCount);
    IF eError <> 0 OR udiCount <> udiSize THEN
        // Обработка ошибок
        EXIT;
    END_IF
    pbySegment := myBuilder.GetNextSegment(pbySegment, udiSize=>udiSize,
       eErrorID=>eErrorID);
END_WHILE

(* Выполнение остального кода – например, закрытие itfConnection *)
```

Работа с ФБ StringPool и RangePool

В примере ниже показано, как использовать динамические экземпляры интерфейса IString с помощью ФБ StringPool и RangePool из библиотеки Generis String Base. Эти блоки хорошо подходят для передачи строк в другие фрагменты приложения. Они позволяют по мере необходимости создавать экземпляры строк из соответствующего пула, работать с ними, а потом возвращать эти экземпляры обратно в пул для освобождения памяти.

```
VAR
    myString : GSB.UTF8String<256> := (sValue:=UTF8#'Was du nicht willst, dass man dir
       tu', das füg auch keinem andern zu.');
    myRange : STR.Range := (itfString:=myString);

    myStringPool : GSB.StringPool<(*udiStringSize*) 30, (*udiInitialCapacity*) 25,
      (*usiExtensionFactor*) 0>;
    myRangePool : GSB.RangePool<GSB.RANGE_TYPE.ISTRING, (*udiInitialCapacity*) 10,
       (*usiExtensionFactor*) 0>;

    diRune : STR.RUNE;
    eErrorID : STR.ERROR;
    itfSubString : STR.IString;
    liStart, liEnd : LINT;
    udiCount : UDINT;
END_VAR

myRange.Reset();
// Разбиваем на подстроки для дальнейшего анализа
WHILE (diRune:=myRange.GetNextRune(eErrorID=>eErrorID)) <> 0 AND eErrorID = 0 DO
    IF diRune = 16#2C (*,*) OR diRune = 16#2E (*.*) THEN
         itfSubString := myStringPool.GetString();
        IF  itfSubString = 0 THEN
            (* Обработка ошибок *)
            EXIT;
        END_IF
        myString.ToIString(itfSubString, liStart+1, liEnd, eErrorID=>eErrorID);
        IF eErrorID <> 0 THEN
            (* Обработка ошибок *)
            EXIT;
        END_IF
        // Анализ подстроки с использованием пула
        // Приведет к очистке itfSubString
        udiCount := Analyse(itfSubString, myStringPool, myRangePool);
        (* ... обработка результата анализа ... *)
        IF diRune = 16#2E (*.*) THEN
            EXIT;
        END_IF
        diRune:=myRange.GetNextRune(eErrorID=>eErrorID);
        IF diRune = 16#20 (* space *) AND eErrorID = 0 THEN
            liEnd := liEnd + 1;
        ELSE
            myRange.UngetLastRune();
        END_IF
        liStart := liEnd + 1;
    END_IF
    liEnd := liEnd + 1;
END_WHILE
```

Определение категории Unicode-символа

Целью стандарта Unicode является объединение всех существующих символов. Символы Unicode разбиты на категории. В состав библиотеки Unicode Support входят функции, позволяющие проверить принадлежность символа к заданной категории. Функции возвращают **TRUE**, если символ принадлежит к категории, и **FALSE** – если не принадлежит.

Функция	Проверяет, является ли символ…
IsControl	…основным управляющим символом
IsLetter	…буквой
IsMark	…составным символом (например, диакритическим знаком)
IsDigit	…десятичной цифрой
IsLower	…символом нижнего регистра
IsNumber	…цифрой в широком смысле этого слова (римской цифрой и т. д.)
IsGraphic	…печатным символом (включая различные виды пробелов)
IsUpper	…верхнего верхнего регистра
IsPunct	…знаком препинания
IsPrint	…печатным символом (включая только один вид пробела с кодом 16#20)
IsTitle	…составным символом, начинающегося с заглавной буквы
IsSpace	…пробелом, концом строки т. д.
IsSymbol	…символом в широком смысле этого слова (математическим символом, символом валюты и т. д.)

С помощью упомянутых функций и ФБ типа «Range» (в разных строковых библиотеках есть разные ФБ с таким названием) можно посимвольно анализировать содержимое экземпляра интерфейса **IString** или **IBuilder**.

```
VAR
    myString : GSB.UTF8String<50> := (sValue:='Hello World!');
    myBuilder : GSB.Builder<100, 0> := (itfString:=myString);
    mySRange : STR.Range := (itfString:=myString);
    myBRange : SBD.Range := (itfBuilder:=myBuilder);
    diSRune, diBRune : STR.RUNE;
    eErrorID : STR.ERROR;
    udiCount : UDINT;
END_VAR

WHILE (diSRune:=mySRange.GetNextRune(eErrorID=>eErrorID)) <> 0 AND eErrorID = 0 DO
    diBRune := myBRange.GetNextRune();
    IF diSRune <> diBRune THEN
        (* это условие не должно выполняться*)
    END_IF
    IF UC.IsSpace(diSrune) THEN
        udiCount := udiCount + 1;
    END_IF
END_WHILE
```

Преобразование регистра символов

```
diRuneA := 16#1F3; // U+01F3 = ǳ

// Преобразование к верхнему регистру
diRuneB := UC.ToUpper(diRuneA); // U+01F1 = Ǳ
// Преобразование к нижнему регистру
diRuneA := UC.ToLower(diRuneB); // U+01F3 = ǳ
// Преобразование составного символа к виду «с заглавной буквы»
diRuneB := UC.ToTitle(diRuneA); // U+01F2 = ǲ
```

Сравнение строк

```
VAR
     myFirstString  : GSB.UTF8String<50> := (sValue:='test');
     mySecondString : GSB.UTF8String<50> := (sValue:='Test');

     myThirdString  : GSB.UTF8String<50> := (sValue:='CoDeSys');
     myFourthString : GSB.UTF8String<50> := (sValue:='CODESYS');

     diResult : DINT;
     xEqual : BOOL;
END_VAR

// Лексикографическое сравнение строк
// diResult = 1 --> myFirstString > mySecondString
diResult := STR.Compare(myFirstString, mySecondString);

// Сравнение строк без учета регистра
// xEqual = TRUE --> myThirdString == myFourthString
xEqual := UC.EqualFold(
    ADR(myThirdString.sValue), myThirdString.Len(),
    ADR(myFourthString.sValue), myFourthString.Len()
);
```