

Clarion SystemString Class API Reference

User Guide 1.0

Contents

1. Help File Layout 8
   1.1. What is SystemString? 8
   1.2. Adding SystemString to your Clarion Application 10
   1.3. State, lifetime and ownership 12
   1.4. Classes and integration 12
   1.5. Workflow examples 13
   1.6. Known issues and verification limits 14
   1.7. About the Author 14
2. Create, assign and clear 14
   2.1. Clean 15
   2.2. FromString 15
   2.3. NewString 17
   2.4. SetString 17
   2.5. Str 19
3. Read values and manage references 20
   3.1. Externalize 20
   3.2. GetString 21
   3.3. GetStringRef 22
   3.4. IsExternalized 23
   3.5. NewStringRef 23
   3.6. ToCString 24
   3.7. ToString 25
4. Build and edit strings 26
   4.1. AddInsertInContentSegment 26
   4.2. AddReplaceInContentSegment 27
   4.3. Append 28
   4.4. AppendLine 30

4.5. CleanSegments 30
4.6. Insert 31
4.7. InsertInContentSegment 32
4.8. Overwrite 33
4.9. Prepend 34
4.10. Remove 34
4.11. RemoveSubString 36
4.12. RemoveSubStringInContent 37
4.13. Replace 38
4.14. ReplaceInContent 38
4.15. ReplaceInContentSegment 40
4.16. ReplaceSubString 41
4.17. SetSubstring 42
4.18. Surround 43
5. Extract portions 44
5.1. GetLeftOf 44
5.2. GetRightOf 45
5.3. GetSurrounded 45
5.4. Substring 46
5.5. SubstringLeft 47
6. Search and compare 48
6.1. Contains 49
6.2. EndsWith 50
6.3. Equals 51
6.4. GetCaseSensitive 53
6.5. IndexOf 53
6.6. IndexOfReg 54
6.7. LastIndexOf 56
6.8. SetCaseSensitive 57

6.9. StartsWith 58
7. Split, edit and join lines 59
7.1. AddLine 60
7.2. CleanLines 60
7.3. CountLines 61
7.4. DeleteLine 62
7.5. FromLines 63
7.6. FromLinesWithEOL 65
7.7. GetEOLMarker 65
7.8. GetLine 66
7.9. GetLines 67
7.10. GetLinesCount 68
7.11. GetLineTrimValue 68
7.12. GetLineValue 69
7.13. InsertLine 70
7.14. Merge 71
7.15. SetEOLMarker 72
7.16. SetLineValue 73
7.17. Split 74
7.18. SplitToLines 75
7.19. TryGetLineValue 76
8. Tokens and words 77
8.1. CountToken 77
8.2. CountTokens 78
8.3. CountWord 79
8.4. CountWords 79
8.5. FoundToken 80
8.6. GetCurrentToken 81
8.7. GetToken 82

8.8. GetTokensDelimiters 83
8.9. GetUseLowerAsciiAsDelimiters 83
8.10. GetWord 84
8.11. IsPastEOL 85
8.12. IsTokenEmpty 86
8.13. IsTokenInited 86
8.14. IsTokenNull 87
8.15. NextToken 88
8.16. PopToken 89
8.17. PushToken 90
8.18. ResetToken 91
8.19. SetTokensDelimiters 91
8.20. SetUseLowerAsciiAsDelimiters 92
9. Change case, trim and pad 93
9.1. PadCenter 93
9.2. PadLeft 94
9.3. PadRight 96
9.4. ToCapitalize 97
9.5. ToCapitalizeInContext 98
9.6. ToLower 98
9.7. ToLowerInContext 100
9.8. ToUpper 100
9.9. ToUpperInContext 102
9.10. Trim 102
9.11. TrimEnd 104
9.12. TrimInContext 105
9.13. TrimStart 106
10. Format and process key/value pairs 107
10.1. ExtractKeyValue 108

10.2. FormatString 109
10.3. GetPairSeparator 110
10.4. GetValueSeparator 111
10.5. ReplaceKeyValue 111
10.6. SetPairSeparator 113
10.7. SetPairValueSeparator 114
10.8. SetValueSeparator 115
11. Encode, decode and compress 115
11.1. Compress 116
11.2. DecodeBase64 117
11.3. Decompress 118
11.4. EncodeBase64 119
11.5. Escape 120
11.6. FromBase64 121
11.7. GetUrlEncodeDataChar 122
11.8. HtmlEscape 122
11.9. SetUrlEncodeDataChar 123
11.10. ToBase64 124
11.11. UnEscape 125
11.12. UrlDecode 126
11.13. UrlEncode 127
12. Files and BLOBs 128
12.1. FromBlob 128
12.2. FromFile 129
12.3. ToBlob 130
12.4. ToFile 131
13. Capacity and append position 132
13.1. GetAppendOnCursor 132
13.2. GetClipOnCursor 133

13.3. GetCursor 134
13.4. Length 135
13.5. Resize 135
13.6. SetAppendOnCursor 136
13.7. SetClipOnCursor 137
13.8. SetCursor 138
13.9. SetLen 139
13.10. SetLength 139
14. Lifecycle and implementation details 140
14.1. StringRefFactoryClass.Construct 141
14.2. StringRefFactoryClass.Destruct 141
14.3. StringRefFactoryClass.DisposeIt 142
14.4. StringRefFactoryClass.NewStringRef 143
14.5. AutoDispose 144
14.6. CleanSplitQueue 145
14.7. Construct 145
14.8. Destruct 146
14.9. DisposeIt 147
14.10. DisposeItCString 148
14.11. GetTrimmingLimits 148
14.12. Internalize 150
14.13. IsDelimiter 150
14.14. IsEOLMarker 151
14.15. Take24 152
14.16. Take32 153
15. Alphabetical API index 154
16. Categories overview 157

1. Help File Layout

New to SystemString?

What is SystemString?
Adding SystemString to your Clarion Application
Browse by task using the categories in Contents, or use the alphabetical index to find a method by name. Method pages show declarations, parameters, returns and setup instructions with sample input and expected output for each overload. Technical notes and implementation evidence are available through a separate link. Based on the supplied SystemString(3).inc and SystemString(3).clw. All behavior is source-derived; Clarion examples and runtime-dependent behavior remain unverified.

Create, assign and clear
Read values and manage references
Build and edit strings
Extract portions
Search and compare
Split, edit and join lines
Tokens and words
Change case, trim and pad
Format and process key/value pairs
Encode, decode and compress
Files and BLOBs
Capacity and append position
Lifecycle and implementation details

Reference

Ownership
Types
Examples
Known-issues
API

1.1. What is SystemString?

SystemString is a source-based string utility supplied with Clarion. Its main class, SystemStringClass, combines a dynamically allocated string with methods for editing, searching, splitting, formatting and encoding text. The SystemString.inc file declares the classes and methods; SystemString.clw contains their implementation. SoftVelocity appears to have introduced SystemString with Clarion 9.0

Why use it?

A SystemStringClass object lets you load text once and then work with it through named methods. You can append text, find a substring, split text into lines, walk through tokens, change case, format values, or encode data. The class manages its own main string allocation; some methods also create queues, caches or temporary buffers.

A small example

```
! Procedure Local Data
Text SystemStringClass
```

```
Result STRING(80)

 CODE
Text.Str('Hello Clarion')
Result = Text.ToUpper()
MESSAGE(CLIP(Result))
```

Expected result

The message displays "HELLO CLARION". Text.Str() still contains "Hello Clarion": ToUpper returns a converted value without assigning it to the object. ToUpperInContext would change the stored string instead.

Why this help file was created

The supplied source contains declarations, comments and implementation code, but it does not provide a complete task-oriented guide to using every method. A declaration alone may not tell you what must be set up first, whether a result is returned or stored, who owns a reference, or which overload to choose. Some source comments also disagree with the implementation.

This reference brings those details together: categories for finding a method, complete declarations, parameter and return descriptions, setup instructions, examples and expected results. Known limitations are called out rather than silently assuming the behavior suggested by a method name. Linked technical notes retain the source evidence behind each description.

Three things to understand first

Returned text and stored text are different. A method such as Trim or ToUpper returns a value; a method such as TrimInContext or ToUpperInContext changes the object. Check the method page before assuming that a call changes its input.

Line records and token selection are additional state. Loading a string does not automatically split it into lines. Line edits normally need Merge or FromLines to rebuild the main string. Token traversal may require ResetToken and NextToken.

References have owners. A reference returned by GetStringRef, GetLine or ToCString is normally borrowed. Reallocation or cleanup can invalidate it; do not dispose a borrowed reference. Externalize has special ownership rules explained on its own page.

Scope of this reference

This edition describes the supplied SystemString(3).inc and SystemString(3).clw, covering SystemStringClass and StringRefFactoryClass. Other Clarion versions may differ. Expected behavior is derived from those files; documentation examples have not all been compiled or run. This is an independent reference, not a statement of SoftVelocity’s intended behavior or an assurance that a method is defect-free.

Next steps

Adding SystemString to your Clarion Application
Browse functions by category
State, lifetime and ownership
Known issues and verification limits

1.2. Adding SystemString to your Clarion Application

These instructions are for a conventional Clarion Win32 application built from an .APP, with a single EXE as the starting point. The .APP holds application definitions and embeds; the associated project controls what the compiler builds. Menu names and embed captions can vary between Clarion versions and template families.

1. Locate the matching source files

Look in your selected Clarion installation's LibSrc\Win folder for SystemString.inc and SystemString.clw. Use the two files from the same version. This reference includes the supplied files in its Source folder for comparison; do not overwrite your installed Clarion files simply to follow these instructions.

The supplied pair was checked against the files in the authoring machine's Clarion 12 LibSrc\Win folder and matched exactly. Availability in another Clarion installation must be checked locally.

2. Open your .APP and check the application name

Open the .APP in Clarion and select the intended Clarion version for its project. For a new test application, use a name such as MyStringDemo.APP.

Avoid a generated application source file named SystemString.clw: it has the same name as the class implementation. If your application already generates that filename, resolve the naming conflict before adding the class. Renaming only one supplied file is insufficient because the INC's MODULE/LINK attributes and the CLW's INCLUDE refer to the original names.

3. Make the class declaration available

Open Global Embeds. Add the following line in a global include/declaration embed, normally "After Global Includes", outside the global MAP. It must appear before any declaration that uses SystemStringClass. The INC declares classes; it is not a procedure-prototype include for Inside the Global Map.

```
INCLUDE('SystemString.inc'),ONCE
```

Use the installed file through Clarion's configured source search path. If the compiler cannot find it, correct that path or place the matching pair in a dedicated project source folder included in the search path. Merely listing the INC in Solution Explorer does not insert its declarations into generated code.

4. Ensure the implementation is compiled once

The supplied class declaration already contains the following conditional source link:

```
MODULE('SystemString.CLW'),LINK('SystemString.CLW', _ABCLinkMode_),DLL(_ABCDllMode_)
```

For the usual single-EXE ABC source-link arrangement, let the templates manage these defines. The single-EXE project inspected for this reference uses _ABCLinkMode_ = 1 and _ABCDllMode_ = 0. With source linking enabled, the LINK attribute requests SystemString.clw; an extra manual compile entry is normally unnecessary.

If your project arrangement requires an explicit external source entry, select the project in Solution Explorer, right-click and choose Add > Existing Item. Select SystemString.clw from the intended source folder. Confirm that it is treated as a source file to compile, not as a library or resource. Keep only one effective compilation of the class implementation; do not add a second copy from another folder.

An Application Generator alternative is Application > Insert Module, then External Source. Select SystemString.clw as the module file and leave Map Include File blank: do not put SystemString.inc in that field,

since the class declaration is already included outside MAP in step 3. Save the .APP and regenerate. Use either the needed project entry or the external module entry, rather than adding duplicates.

Do not INCLUDE SystemString.clw as text in a procedure or global embed. It is a separate MEMBER source module. Do not change the application-wide ABC defines merely to make one class compile: those settings also affect other ABC classes. Multi-DLL, Legacy-template and custom library arrangements need their own ownership/link configuration and are outside this single-EXE walkthrough.

5. Declare an object in a window procedure

Open a window procedure’s Embeds. In its Local Data declaration embed, add:

```
Text SystemStringClass
Result STRING(80)
```

Use the procedure’s real declaration embed, before executable CODE and outside the WINDOW structure. A local object runs its constructor automatically; it does not need NEW or a manual Construct call.

6. Add a small button test

Add a button to the window. In that button’s Accepted embed, enter:

```
Text.Str('Hello Clarion')
Result = Text.ToUpper()
MESSAGE(CLIP(Result))
```

Expected result: the message displays "HELLO CLARION". The stored string remains "Hello Clarion". This verifies declaration visibility, linkage and a basic method call without requiring a dictionary or file.

7. Save, generate, build and run

Save the .APP and project. Generate the application, build it and run the button test. If you added an external file through Solution Explorer, regenerate again and confirm that the source entry is still present. Keep permanent changes in the .APP embeds or supported project settings; edits made directly to generated .CLW files can be lost at the next generation.

If it does not build

Unknown type SystemStringClass: check the INC name, search path and include placement before the object declaration. Check that you have not placed it inside MAP.

Unresolved SystemStringClass methods: check that the intended SystemString.clw is linked or compiled and that the project’s ABC source/DLL configuration matches the build arrangement.

Duplicate symbols or unexpected methods: check for two compiled copies, a prelinked class plus a source copy, mixed versions, or an application-generated SystemString.clw hiding the class file.

File-method dependencies: the supplied implementation uses the DOS file driver. When using ToFile or FromFile, ensure that DOS is available in the project; Clarion provides Project > Add > File Driver for driver dependencies. The basic uppercase test does not exercise file I/O.

1.3. State, lifetime and ownership

Main value
Main storage s, cached CSTRING cs, scratch garbageStr, line queue and pending segments are separate. Copying/changing main content generally leaves existing lines intact. Split rebuilds lines; FromLines/Merge rebuild main content. Token selection is mutable state.

Borrowed references
GetStringRef, ToCString, GetLine and GetLines return borrowed storage. Do not dispose it or retain it past the operation that replaces/deletes it. NewStringRef returns scratch storage, not main content. StringRefFactoryClass tracks multiple allocations until DisposeIt/destruction.

Transferred reference
Externalize returns main storage to the caller, but implementation suppresses one DisposeIt only. The object still holds the reference. Safest intended pattern is Externalize immediately before ending the owner lifetime, then caller disposal after owner destruction; this lifecycle requires runtime verification.

Index and length rules
String and queue positions are generally 1-based. BLOB positions are zero-based. Length reports allocated length; Str may expose only cursor prefix. Positive valid indices and nonempty separators/needles are caller preconditions unless a topic states otherwise. No general bounds/error framework exists.

1.4. Classes and integration

Step-by-step: Adding SystemString to your Clarion Application

Integration
The INC declares MODULE/LINK SystemString.CLW and ABC DLL/link-mode macros. Include SystemString.inc within the application structure appropriate to your Clarion version; link the supplied CLW exactly once or use the applicable ABC library arrangement. Exact Clarion version and dependent runtime were not supplied. Do not duplicate an already-linked ABC class.

SystemStringQueue
QUEUE,TYPE containing Str &SystemStringClass. Parent-owned line queue normally owns the pointed-to objects.

SystemStringSegmentsQueue
QUEUE,TYPE: startSeg ULONG; endSeg ULONG; Str &STRING. Stores copied replacement bytes and original inclusive positions; endSeg zero denotes insertion.

StringRefFactoryClass
Separate allocation owner. Construct creates StringRefQueue; NewStringRef adds a NEW STRING to it; DisposeIt disposes all entries and FREEs records; Destruct also disposes queue. Its declaration comment incorrectly implies previous contents are immediately disposed.

Module helpers
SystemStringClass_ByteToHex returns two uppercase/lowercase hexadecimal digits. SystemStringClass_IsXDigit actually accepts 0..9, A..Z, a..z rather than only hexadecimal letters. Private alphabet/constants are implementation details.

1.5. Workflow examples

Reading the method examples
Each overload has a Before you call explanation, local declarations, a code fragment and an expected result. Treat separate overload examples as independent runs with new local objects. Place declarations in procedure Local Data and statements in CODE or the appropriate executable embed. Include SystemString.inc and link SystemString.clw using the integration described under Classes and integration.
Result is a fixed STRING(1024), so assigning shorter text pads the variable with spaces. Quoted expected text describes the returned logical value, not all 1024 destination bytes. Use CLIP(Result) for ordinary display; preserve the stated byte count when trailing spaces are significant. Number holds a count, position, flag or status as stated on the page. No MESSAGE dialog or UI control is required by these fragments.
Spaces inside quoted expected values are significant. <13,10> and <9> inside Clarion string literals represent actual control bytes. A backslash in a Clarion literal is a literal backslash; doubled opening braces represent a literal opening brace. For protected methods, SELF denotes a derived object; private methods are demonstrated through accessible public calls.
The supplied implementation is the authority for the expected results. These new documentation snippets have not been compiled or executed. File/BLOB examples require application resources; compression and regular-expression outcomes depend on installed runtime support. Known defects remain flagged on method pages and in the linked technical notes.

Status
Illustrative code snippets, not compiled or executed. Place declarations in local data and statements in CODE or an appropriate embed. Do not manually call Construct/Destruct.

Clarion example
```
! Local data
Text SystemStringClass
Value SystemStringClass
Result STRING(100)
Status LONG
! CODE
Text.Str('Alpha,beta')
Status = Text.IndexOf('beta') ! source predicts 7
Result = Text.ToUpper() ! main content unchanged
Text.ToUpperInContext() ! main content changes
Text.Split(',')
Text.SetLineValue(2,'gamma')
Text.Merge('|') ! source predicts ALPHA|gamma
Text.Str('name=Alice&mode=fast')
IF Text.ExtractKeyValue('name',Value)
 Result = Value.Str() ! source predicts Alice
END
```

Returned value versus mutation
Result = Text.Insert(1,'prefix:') leaves Text unchanged. Text.Str(Text.Insert(1,'prefix:')) assigns the result. Prefer Overwrite for exact-end writes; SetSubstring has a stricter boundary check. Keep file status tests as status=0, not IF status.

Ownership example
Declare Borrowed &STRING, assign Borrowed &= Text.GetStringRef(), and use only while Text retains the allocation. Do not DISPOSE Borrowed. A subsequent Text.Str(...) invalidates it. Prefer copying Str() when lifetime is uncertain.

1.6. Known issues and verification limits

Source findings
Read each linked method before relying on nominal names. High-impact findings include one-use externalization suppression; ignored pStr arguments; lost single-byte replacement/HTML tails; strict positional replacement guards; right-only in-context trim skipped; malformed URL lowercase decode; Base64 result not shrunk; one-line Merge retaining delimiter; token boundary using start+1.

Unverified behavior
Allocation failures, NULL references, invalid slices, omitted-argument forwarding, Clarion STRING padding/comparison, STRPOS regex syntax, locale case conversion, DOS error constants, compression interoperability and compiler-version behavior were not executed. No runtime test results are claimed.

Suggested regression cases
Test empty/null, single-byte, all-space, embedded NUL, final-character match, exact-end overwrite, absent needle, empty delimiter rejection, adjacent/trailing split separators, one-line merge, control-character tokens, escaped backslash pairs, lowercase percent escapes, Base64 padding and binary data, and Externalize followed by multiple owner operations.

1.7. About the Author

This help file was created by Software by Daughtry, using ChatGPT to analyze the SystemString source files and then generate a categorized help file. Questions regarding this user manual may be sent to:

scott@sdaughtry.com

http://www.sdaughtry.com

This .CHM help file (and its associated .PDF manual) are released free to the Clarion community, and may not be offered for sale.

SoftVelocity is the sole owner of SystemString, and all rights to SystemString belong to SoftVelocity.

2. Create, assign and clear

Select a method below. Detailed source evidence is available through each method's Technical notes link.
Clean
FromString
NewString
SetString
Str

2.1. Clean

```
Clean PROCEDURE()
```

Description

Sets main content to empty and disposes all line objects.

Parameters

None.

Returns

No return value.

Before you call

Populate both main text and the separate line queue to see both effects.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('red,blue')
Text.Split(',')
Text.Clean()
Result = Text.Str()
Number = Text.CountLines()
```

Expected result

Result is empty and line count is 0. Clean is not a reset of every configuration setting.

See also

Back to category
Technical notes and source evidence
SetString
CleanLines

2.2. FromString

```
FromString PROCEDURE(STRING svalue)
FromString PROCEDURE(*CSTRING svalue)
```

Description

Setter overloads copy supplied content into newly allocated main storage, resetting token selection and externalization state; SetString sets cursor to input LEN+1.

Parameters

Parameter	Data type	Description
svalue	STRING	Input text.
svalue	*CSTRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.
Assign the CSTRING input first. The method copies its text; it does not take ownership of the CSTRING.

Example 1 — FromString(STRING svalue)

```
! Local Data
Text SystemStringClass
Result STRING(1024)
InputC CSTRING(32)

 CODE
Text.Str('old')
Text.FromString('cat')
Result = Text.Str()
```

Expected result

Stored string: "cat". This call has no return value; Result reads the stored string afterward.

Example 2 — FromString(*CSTRING svalue)

```
InputC = 'cat'
Text.FromString(InputC)
Result = Text.Str()
```

Expected result

Stored string and Result are "cat". InputC remains owned by the caller.

2.3. NewString

```
NewString PROCEDURE(ULONG sizeValue=0)
```

Description

Disposes/replaces main storage with NEW STRING(sizeValue), resets externalization flag and cursor to 1.

Parameters

Parameter	Data type	Description
sizeValue=0	ULONG	Requested byte length/count or width. See allocation and range restrictions.

Returns

No return value.

Before you call

NewString allocates replacement storage; loading a previous value is optional.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('cat')
Text.NewString(5)
Number = Text.Length()
Result = Text.Str()
```

Expected result

Number is 5; Result contains five spaces. Previous content is discarded and cursor becomes 1.

2.4. SetString

```
SetString PROCEDURE(STRING svalue)
SetString PROCEDURE(*CSTRING svalue)
```

Description

Setter overloads copy supplied content into newly allocated main storage, resetting token selection and externalization state; SetString sets cursor to input LEN+1.

Parameters

Parameter	Data type	Description
svalue	STRING	Input text.
svalue	*CSTRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.
Assign the CSTRING input first. The method copies its text; it does not take ownership of the CSTRING.

Example 1 — SetString(STRING svalue)

```
! Local Data
Text SystemStringClass
Result STRING(1024)
InputC CSTRING(32)

 CODE
Text.Str('old')
Text.SetString('cat')
Result = Text.Str()
```

Expected result

Stored string: "cat". This call has no return value; Result reads the stored string afterward.

Example 2 — SetString(*CSTRING svalue)

```
InputC = 'cat'
Text.SetString(InputC)
Result = Text.Str()
```

Expected result

Stored string and Result are "cat". InputC remains owned by the caller.

2.5. Str

```
Str PROCEDURE(STRING svalue)
Str PROCEDURE(SystemStringClass s)
Str PROCEDURE(),STRING
```

Description

Setter overloads copy supplied content into newly allocated main storage, resetting token selection and externalization state; SetString sets cursor to input LEN+1.

Parameters

Parameter	Data type	Description
svalue	STRING	Input text.
s	SystemStringClass	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.
STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.
Load the source object; Str copies its value rather than sharing its allocation.

Example 1 — Str(STRING svalue)

```
! Local Data
Text SystemStringClass
Other SystemStringClass
Result STRING(1024)

 CODE
Text.Str('old')
Text.Str('cat')
Result = Text.Str()
```

Expected result

Stored string: "cat". This call has no return value; Result reads the stored string afterward.

Example 2 — Str(SystemStringClass s)

```
Other.Str('cat')
```

```
Text.Str(Other)
Result = Text.Str()
```

Expected result

Text and Other both contain "cat" in separate owned allocations.

Example 3 — Str(),STRING

```
Text.Str('cat')
Result = Text.Str()
```

Expected result

Returned text: "cat".
Stored string: "cat" (unchanged).

3. Read values and manage references

Select a method below. Detailed source evidence is available through each method's Technical notes link.
Externalize
GetString
GetStringRef
IsExternalized
NewStringRef
ToCString
ToString

3.1. Externalize

```
Externalize PROCEDURE(),*STRING
```

Description

Returns the main STRING reference and sets alreadyDisposed and wasExternalized true.

Parameters

None.

Returns

Main STRING reference, transferred subject to the lifecycle warning below.

Before you call

Use a separately allocated owner, transfer once, destroy that owner once, then dispose the transferred string. See the ownership warning.

Example

```
! Local Data
Result STRING(1024)
Ref &STRING
Owner &SystemStringClass

 CODE
Owner &= NEW SystemStringClass
Owner.Str('cat')
Ref &= Owner.Externalize()
DISPOSE(Owner)
Result = Ref
DISPOSE(Ref)
```

Expected result

Result contains "cat". The one disposal suppression lets the string survive destruction of this owner; the caller then releases it.
Do not call DisposeIt before DISPOSE(Owner) in this sequence: that would consume the one suppression.

Notes

Ownership transfer suppresses only one disposal call. Follow the lifecycle notes before using this method.

3.2. GetString

```
GetString PROCEDURE(),STRING
```

Description

Returns Str() by value.

Parameters

None.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('cat')
```

```
Result = Text.GetString()
```

Expected result

Returned text: "cat".
Stored string: "cat" (unchanged).

3.3. GetStringRef

```
GetStringRef PROCEDURE(),*STRING
```

Description

Returns a borrowed reference to main storage, possibly NULL.

Parameters

None.

Returns

Borrowed reference to main STRING storage, possibly NULL.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Ref &STRING

 CODE
Text.Str('cat')
Ref &= Text.GetStringRef()
Ref[1] = 'b'
Result = Text.Str()
```

Expected result

Borrowed Ref accesses the main allocation. Stored string and Result become "bat". Do not DISPOSE Ref.

3.4. IsExternalized

```
IsExternalized PROCEDURE(),BYTE
```

Description

Returns wasExternalized.

Parameters

None.

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

A new object starts with IsExternalized FALSE. No transfer is needed merely to query it.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('cat')
Number = Text.IsExternalized()
```

Expected result

Returned value: 0 (FALSE). Stored string remains "cat". Externalize changes this flag, but it is not a permanent ownership guarantee.

3.5. NewStringRef

```
NewStringRef PROCEDURE(ULONG sizeValue),*STRING
```

Description

SystemStringClass allocates scratch storage, not main storage: disposes previous garbageStr, allocates STRING(sizeValue), returns its reference.

Parameters

Parameter	Data type	Description
sizeValue	ULONG	Requested byte length/count or width. See allocation and range restrictions.

Returns

Borrowed scratch STRING reference; next scratch allocation invalidates it.

Before you call

Load text only to demonstrate that scratch storage is separate from the main string.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Ref &STRING

 CODE
Text.Str('cat')
Ref &= Text.NewStringRef(3)
Ref = 'dog'
Result = Ref
```

Expected result

Scratch buffer contains "dog"; Text.Str() remains "cat". Ref is owned by Text and becomes invalid at the next scratch allocation or destruction.

Notes

Returns scratch storage, not the main string. The next scratch allocation invalidates the reference.

3.6. ToCString

```
ToCString PROCEDURE(),*CSTRING
```

Description

Disposes the previous cached CSTRING, creates CSTRING(LEN(CLIP(s))+1), assigns CLIP(s), and returns a borrowed reference.

Parameters

None.

Returns

Borrowed CSTRING reference; invalidated by the next ToCString or cache disposal.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
CRef &CSTRING

 CODE
Text.Str('cat ')
CRef &= Text.ToCString()
Result = CRef
```

Expected result

Returned CSTRING text: "cat" (trailing spaces clipped). Stored string remains "cat ". Do not dispose CRef.

3.7. ToString

```
ToString PROCEDURE(),STRING
```

Description

Returns Str() by value.

Parameters

None.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
```

```
Text.Str('cat')
Result = Text.ToString()
```

Expected result

Returned text: "cat".
Stored string: "cat" (unchanged).

4. Build and edit strings

Select a method below. Detailed source evidence is available through each method's Technical notes link.

AddInsertInContentSegment
AddReplaceInContentSegment
Append
AppendLine
CleanSegments
Insert
InsertInContentSegment
Overwrite
Prepend
Remove
RemoveSubString
RemoveSubStringInContent
Replace
ReplaceInContent
ReplaceInContentSegment
ReplaceSubString
SetSubstring
Surround

4.1. AddInsertInContentSegment

```
AddInsertInContentSegment PROCEDURE(LONG insertIndex, STRING svalue)
```

Description

Queues copied replacement bytes in an owned segments queue.

Parameters

Parameter	Data type	Description
insertIndex	LONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.
svalue	STRING	Input text.

Returns

No return value.

Before you call

Load text, queue an insertion with AddInsertInContentSegment, then call InsertInContentSegment to apply it. Positions refer to the original source string.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('redblue')
Text.AddInsertInContentSegment(4, ',')
! Still "redblue" here: the insertion is only queued.
Text.InsertInContentSegment()
Result = Text.Str()
```

Expected result

Stored string after applying: "red,blue". The pending segment queue is cleaned after application.

4.2. AddReplaceInContentSegment

```
AddReplaceInContentSegment PROCEDURE(LONG startSeg, LONG endSeg, STRING svalue)
```

Description

Queues copied replacement bytes in an owned segments queue.

Parameters

Parameter	Data type	Description
startSeg	LONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.
endSeg	LONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.
svalue	STRING	Input text.

Returns

No return value.

Before you call

Load text, queue a replacement with inclusive original-source endpoints, then apply it. Use nonempty replacement text and nonoverlapping valid ranges.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('red,blue')
Text.AddReplaceInContentSegment(1, 3, 'tan')
! Still "red,blue" here: the replacement is only queued.
Text.ReplaceInContentSegment()
Result = Text.Str()
```

Expected result

Stored string after applying: "tan,blue". Equal-length replacement avoids the builder’s trailing-capacity issue.

4.3. Append

```
Append PROCEDURE(STRING svalue, BYTE pAddEOLMarker=0)
Append PROCEDURE(SystemStringClass svalue)
Append PROCEDURE(STRING leftValue, SystemStringClass svalue, STRING rightValue)
```

Description

Appends input or overwrites from appendCursor when cursor mode is enabled, growing storage as needed.

Parameters

Parameter	Data type	Description
svalue	STRING	Input text.
pAddEOLMarker=0	BYTE	Append configured EOL marker with supplied text when nonzero.
svalue	SystemStringClass	Input text, object, delimiter or configuration value as used in behavior notes.
leftValue	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
rightValue	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.
Load both objects; appending copies the source value.
Load both objects; wrapper strings surround the appended object value.

Example 1 — Append(STRING svalue,BYTE pAddEOLMarker=0)

```
! Local Data
Text SystemStringClass
Other SystemStringClass
Result STRING(1024)

 CODE
Text.Str('Hello')
Text.Append(' world')
Result = Text.Str()
```

Expected result

Stored string: "Hello world". This call has no return value; Result reads the stored string afterward.

Example 2 — Append(SystemStringClass svalue)

```
Text.Str('Hello ')
Other.Str('world')
Text.Append(Other)
Result = Text.Str()
```

Expected result

Stored string: "Hello world". Other remains "world".

Example 3 — Append(STRING leftValue,SystemStringClass svalue,STRING rightValue)

```
Text.Str('Hello ')
Other.Str('world')
Text.Append('[', Other, ']')
Result = Text.Str()
```

Expected result

Stored string: "Hello [world]". Other remains "world".

4.4. AppendLine

```
AppendLine PROCEDURE(STRING svalue)
```

Description

Appends input or overwrites from appendCursor when cursor mode is enabled, growing storage as needed.

Parameters

Parameter	Data type	Description
svalue	STRING	Input text.

Returns

No return value.

Before you call

Load text first. Set an easily visible EOL marker for this example.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.SetEOLMarker('|')
Text.Str('cat|')
Text.AppendLine('dog')
Result = Text.Str()
```

Expected result

Stored string: "cat|dog|". The existing marker after cat was part of the starting text.

4.5. CleanSegments

```
CleanSegments PROCEDURE()
```

Description

Disposes each copied segment string, frees records and disposes the segments queue.

Parameters

None.

Returns

No return value.

Before you call

Queue an edit first to demonstrate cancellation. CleanSegments discards pending edits; it does not apply them.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('redblue')
Text.AddInsertInContentSegment(4, ',')
Text.CleanSegments()
Text.InsertInContentSegment()
Result = Text.Str()
```

Expected result

Stored string remains "redblue": the insertion was discarded before application.

4.6. Insert

```
Insert PROCEDURE(ULONG startIndex, STRING value),STRING
```

Description

Returns a new value with input before startIndex; index <2 prepends, index beyond length appends.

Parameters

Parameter	Data type	Description
startIndex	ULONG	1-based starting position.
value	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('redblue')
Result = Text.Insert(4, ',')
```

Expected result

Returned text: "red,blue".
Stored string: "redblue" (unchanged).

4.7. InsertInContentSegment

```
InsertInContentSegment PROCEDURE()
```

Description

Sorts queued segments by original start index, builds replacement content, applies valid ranges and disposes queued segments.

Parameters

None.

Returns

No return value.

Before you call

Load text, queue an insertion with AddInsertInContentSegment, then call InsertInContentSegment to apply it. Positions refer to the original source string.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('redblue')
Text.AddInsertInContentSegment(4, ',')
! Still "redblue" here: the insertion is only queued.
Text.InsertInContentSegment()
Result = Text.Str()
```

Expected result

Stored string after applying: "red,blue". The pending segment queue is cleaned after application.

4.8. Overwrite

```
Overwrite PROCEDURE(LONG idxReplaceStart, STRING svalue, BYTE resizeIfneeded=FALSE),BYTE,PROC
```

Description

Writes nonempty input directly at positive idxReplaceStart.

Parameters

Parameter	Data type	Description
idxReplaceStart	LONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.
svalue	STRING	Input text.
resizeIfneeded=FALSE	BYTE	Allow main storage to grow if write extends beyond current length.

Returns

BYTE — TRUE if the write was performed; otherwise FALSE.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('red blue')
Number = Text.Overwrite(5, 'gold')
```

Expected result

Returned value: 1.
Stored string: "red gold".

4.9. Prepend

```
Prepend PROCEDURE(STRING svalue, BYTE pAddEOLMarker=0)
```

Description

Prepend places input before existing content and optionally places EOL between input and old value.

Parameters

Parameter	Data type	Description
svalue	STRING	Input text.
pAddEOLMarker=0	BYTE	Append configured EOL marker with supplied text when nonzero.

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('world')
Text.Prepend('Hello ')
Result = Text.Str()
```

Expected result

Stored string: "Hello world". This call has no return value; Result reads the stored string afterward.

4.10. Remove

```
Remove PROCEDURE(ULONG startIndex),STRING
Remove PROCEDURE(ULONG startIndex, ULONG subLength),STRING
```

Description

Returns a STRING; does not modify main storage.

Parameters

Parameter	Data type	Description
startIndex	ULONG	1-based starting position.
subLength	ULONG	Requested byte length/count or width. See allocation and range restrictions.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — Remove(ULONG startIndex),STRING

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('abcdef')
Result = Text.Remove(3)
```

Expected result

Returned text: "ab".
Stored string: "abcdef" (unchanged).

Example 2 — Remove(ULONG startIndex,ULONG subLength),STRING

```
Text.Str('abcdef')
Result = Text.Remove(3, 2)
```

Expected result

Returned text: "abef".
Stored string: "abcdef" (unchanged).

4.11. RemoveSubString

```
RemoveSubString PROCEDURE(STRING originalString, STRING subString),STRING
RemoveSubString PROCEDURE(STRING subString),STRING
```

Description

Returns transformed STRING without assigning main content.

Parameters

Parameter	Data type	Description
originalString	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
subString	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

This overload takes the original text as an argument; no main text is required.
Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — RemoveSubString(STRING originalString,STRING subString),STRING

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Result = Text.RemoveSubString('red blue', 'red ')
```

Expected result

Returned text: "blue". Main string is not assigned.

Example 2 — RemoveSubString(STRING subString),STRING

```
Text.Str('red blue')
Result = Text.RemoveSubString('red ')
```

Expected result

Returned text: "blue".
Stored string: "red blue" (unchanged).

4.12. RemoveSubStringInContent

```
RemoveSubStringInContent PROCEDURE(STRING subString),LONG,PROC
```

Description

Calls ReplaceInContent(subString,empty), mutates main content and returns replacement count.

Parameters

Parameter	Data type	Description
subString	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

LONG — number of replacements applied.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('red blue')
Number = Text.RemoveSubStringInContent('red ')
```

Expected result

Returned value: 1.
Stored string: "blue".

4.13. Replace

```
Replace PROCEDURE(STRING oldString,STRING newString),STRING
```

Description

Returns transformed STRING without assigning main content.

Parameters

Parameter	Data type	Description
oldString	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
newString	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('red blue')
Result = Text.Replace('red', 'tan')
```

Expected result

Returned text: "tan blue".
Stored string: "red blue" (unchanged).

4.14. ReplaceInContent

```
ReplaceInContent PROCEDURE(STRING oldString,STRING newString),LONG,PROC
ReplaceInContent PROCEDURE(LONG idxReplaceStart, LONG strReplaceLen,STRING newString),BYTE,PROC
```

Description

STRING overload replaces nonoverlapping matches in main content, returning count; skips empty oldString, equal old/new, and empty main content.

Parameters

Parameter	Data type	Description
oldString	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
newString	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
idxReplaceStart	LONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.
strReplaceLen	LONG	Requested byte length/count or width. See allocation and range restrictions.

Returns

STRING-search overload: LONG replacement count. Positional overload: BYTE success flag.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — ReplaceInContent(STRING oldString,STRING newString),LONG,PROC

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('red blue')
Number = Text.ReplaceInContent('red', 'tan')
```

Expected result

Returned value: 1.
Stored string: "tan blue".

Example 2 — ReplaceInContent(LONG idxReplaceStart,LONG strReplaceLen,STRING newString),BYTE,PROC

```
Text.Str('red blue')
Number = Text.ReplaceInContent(1, 3, 'tan')
```

Expected result

Returned value: 1.
Stored string: "tan blue".

Notes

The STRING overload can drop a final one-byte tail. The positional overload has restrictive end checks.

4.15. ReplaceInContentSegment

```
ReplaceInContentSegment PROCEDURE()
```

Description

Sorts queued segments by original start index, builds replacement content, applies valid ranges and disposes queued segments.

Parameters

None.

Returns

No return value.

Before you call

Load text, queue a replacement with inclusive original-source endpoints, then apply it. Use nonempty replacement text and nonoverlapping valid ranges.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('red,blue')
Text.AddReplaceInContentSegment(1, 3, 'tan')
! Still "red,blue" here: the replacement is only queued.
Text.ReplaceInContentSegment()
Result = Text.Str()
```

Expected result

Stored string after applying: "tan,blue". Equal-length replacement avoids the builder’s trailing-capacity issue.

4.16. ReplaceSubString

```
ReplaceSubString PROCEDURE(STRING originalString, STRING oldString,STRING newString),STRING
```

Description

Returns transformed STRING without assigning main content.

Parameters

Parameter	Data type	Description
originalString	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
oldString	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
newString	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

The input text is an argument, so loading Text is unnecessary. Its case setting still controls matching.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Result = Text.ReplaceSubString('red blue', 'red', 'tan')
```

Expected result

Returned text: "tan blue". Main string is not assigned.

4.17. SetSubstring

```
SetSubstring PROCEDURE(ULONG startIndex, STRING newString),BYTE,PROC
```

Description

Overwrites LEN(newString) bytes only when startIndex >0 AND startIndex+LEN(newString) < Length().

Parameters

Parameter	Data type	Description
startIndex	ULONG	1-based starting position.
newString	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

BYTE — TRUE if the write was performed; otherwise FALSE.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('red blue')
Number = Text.SetSubstring(1, 'tan')
```

Expected result

Returned value: 1.
Stored string: "tan blue".

Notes

Writes at or near the end are rejected by a stricter check than Overwrite uses.

4.18. Surround

```
Surround PROCEDURE(STRING leftValue, STRING rightValue)
```

Description

Prepend places input before existing content and optionally places EOL between input and old value.

Parameters

Parameter	Data type	Description
leftValue	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
rightValue	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('cat')
Text.Surround('[', ']')
Result = Text.Str()
```

Expected result

Stored string: "[cat]". This call has no return value; Result reads the stored string afterward.

5. Extract portions

Select a method below. Detailed source evidence is available through each method’s Technical notes link.
GetLeftOf
GetRightOf
GetSurrounded
Substring
SubstringLeft

5.1. GetLeftOf

```
GetLeftOf PROCEDURE(STRING svalue),STRING
```

Description

Searches first occurrence using IndexOf.

Parameters

Parameter	Data type	Description
svalue	STRING	Input text.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('red:blue')
Result = Text.GetLeftOf(':')
```

Expected result

Returned text: "red".
Stored string: "red:blue" (unchanged).

5.2. GetRightOf

```
GetRightOf PROCEDURE(STRING svalue),STRING
```

Description

Searches first occurrence using IndexOf.

Parameters

Parameter	Data type	Description
svalue	STRING	Input text.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('red:blue')
Result = Text.GetRightOf(':')
```

Expected result

Returned text: "blue".
Stored string: "red:blue" (unchanged).

5.3. GetSurrounded

```
GetSurrounded PROCEDURE(STRING leftValue, STRING rightValue,ULONG startIndex=1,ULONG endIndex=0),STRING
```

Description

Finds first left marker at/after startIndex, then first right marker after it; returns intervening bytes.

Parameters

Parameter	Data type	Description
leftValue	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
rightValue	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
startIndex=1	ULONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.
endIndex=0	ULONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('red[green]blue')
Result = Text.GetSurrounded('[', ']')
```

Expected result

Returned text: "green".
Stored string: "red[green]blue" (unchanged).

5.4. Substring

```
Substring PROCEDURE(ULONG startIndex),STRING
Substring PROCEDURE(ULONG startIndex, ULONG subLength),STRING
```

Description

Returns suffix or up to subLength bytes starting at startIndex; truncates at end.

Parameters

Parameter	Data type	Description
startIndex	ULONG	1-based starting position.
subLength	ULONG	Requested byte length/count or width. See allocation and range restrictions.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — Substring(ULONG startIndex),STRING

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('abcdef')
Result = Text.Substring(3)
```

Expected result

Returned text: "cdef".
Stored string: "abcdef" (unchanged).

Example 2 — Substring(ULONG startIndex,ULONG subLength),STRING

```
Text.Str('abcdef')
Result = Text.Substring(3, 2)
```

Expected result

Returned text: "cd".
Stored string: "abcdef" (unchanged).

5.5. SubstringLeft

```
SubstringLeft PROCEDURE(ULONG startIndex),STRING
SubstringLeft PROCEDURE(ULONG startIndex, ULONG subLength),STRING
```

Description

Returns bytes immediately before startIndex; excludes startIndex.

Parameters

Parameter	Data type	Description
startIndex	ULONG	1-based starting position.
subLength	ULONG	Requested byte length/count or width. See allocation and range restrictions.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — SubstringLeft(ULONG startIndex),STRING

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('abcdef')
Result = Text.SubstringLeft(4)
```

Expected result

Returned text: "abc".
Stored string: "abcdef" (unchanged).

Example 2 — SubstringLeft(ULONG startIndex,ULONG subLength),STRING

```
Text.Str('abcdef')
Result = Text.SubstringLeft(4, 2)
```

Expected result

Returned text: "bc".
Stored string: "abcdef" (unchanged).

6. Search and compare

Select a method below. Detailed source evidence is available through each method's Technical notes link.
Contains
EndsWith
Equals
GetCaseSensitive
IndexOf

IndexOfReg
LastIndexOf
SetCaseSensitive
StartsWith

6.1. Contains

```
Contains PROCEDURE(STRING svalue,ULONG startIndex=1),BYTE
```

Description

IndexOf searches with INSTRING(...,1,startIndex), using uppercase copies when caseSensitive is false; default start is 1.

Parameters

Parameter	Data type	Description
svalue	STRING	Text to find or compare.
startIndex=1	ULONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.

Returns

BYTE — TRUE if a match exists; otherwise FALSE.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('red blue')
Number = Text.Contains('blue')
```

Expected result

Returned value: 1.
Stored string: "red blue" (unchanged).

6.2. EndsWith

```
EndsWith PROCEDURE(SystemStringClass svalue),BYTE
EndsWith PROCEDURE(SystemStringClass svalue, BYTE IgnoreCase),BYTE
EndsWith PROCEDURE(STRING svalue),BYTE
EndsWith PROCEDURE(STRING svalue,BYTE IgnoreCase),BYTE
```

Description

Returns TRUE/FALSE from prefix, suffix or whole-value comparison.

Parameters

Parameter	Data type	Description
svalue	SystemStringClass	Input text, object, delimiter or configuration value as used in behavior notes.
IgnoreCase	BYTE	TRUE selects case-insensitive comparison; FALSE selects case-sensitive comparison.
svalue	STRING	Text to find or compare.

Returns

BYTE — TRUE if the suffix matches; otherwise FALSE.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically. Load Other as the comparison object.
Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — EndsWith(SystemStringClass svalue),BYTE

```
! Local Data
Text SystemStringClass
Other SystemStringClass
Number LONG

 CODE
Text.Str('Hello world')
Other.Str('world')
Number = Text.EndsWith(Other)
```

Expected result

Returned flag: 1 (TRUE). Stored string remains "Hello world".

Example 2 — EndsWith(SystemStringClass svalue,BYTE IgnoreCase),BYTE

```
Text.Str('Hello world')
Other.Str('world')
Number = Text.EndsWith(Other, FALSE)
```

Expected result

Returned flag: 1 (TRUE). Stored string remains "Hello world".

Example 3 — EndsWith(STRING svalue),BYTE

```
Text.Str('Hello world')
Number = Text.EndsWith('world')
```

Expected result

Returned flag: 1 (TRUE). Stored string remains "Hello world".

Example 4 — EndsWith(STRING svalue,BYTE IgnoreCase),BYTE

```
Text.Str('Hello world')
Number = Text.EndsWith('world', FALSE)
```

Expected result

Returned flag: 1 (TRUE). Stored string remains "Hello world".

6.3. Equals

```
Equals PROCEDURE(SystemStringClass svalue),BYTE
Equals PROCEDURE(STRING svalue),BYTE
Equals PROCEDURE(STRING svalue, BYTE IgnoreCase),BYTE
```

Description

Returns TRUE/FALSE from prefix, suffix or whole-value comparison.

Parameters

Parameter	Data type	Description
svalue	SystemStringClass	Input text, object, delimiter or configuration value as used in behavior notes.
svalue	STRING	Text to find or compare.

IgnoreCase	BYTE	TRUE selects case-insensitive comparison; FALSE selects case-sensitive comparison.

Returns

BYTE — TRUE if the values compare equal; otherwise FALSE.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically. Load Other as the comparison object.
Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — Equals(SystemStringClass svalue),BYTE

```
! Local Data
Text SystemStringClass
Other SystemStringClass
Number LONG

 CODE
Text.Str('Hello world')
Other.Str('Hello world')
Number = Text.Equals(Other)
```

Expected result

Returned flag: 1 (TRUE). Stored string remains "Hello world".

Example 2 — Equals(STRING svalue),BYTE

```
Text.Str('Hello world')
Number = Text.Equals('Hello world')
```

Expected result

Returned flag: 1 (TRUE). Stored string remains "Hello world".

Example 3 — Equals(STRING svalue,BYTE IgnoreCase),BYTE

```
Text.Str('Hello world')
Number = Text.Equals('Hello world', FALSE)
```

Expected result

Returned flag: 1 (TRUE). Stored string remains "Hello world".

6.4. GetCaseSensitive

```
GetCaseSensitive PROCEDURE(),BYTE
```

Description

Returns the corresponding stored BYTE configuration flag.

Parameters

None.

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

No string needs to be loaded to read or change this setting. Configure it before the operation that uses it.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Number = Text.GetCaseSensitive()
```

Expected result

New-object result: 1 (TRUE).

6.5. IndexOf

```
IndexOf PROCEDURE(STRING svalue),LONG
IndexOf PROCEDURE(STRING svalue, ULONG startIndex),LONG
```

Description

IndexOf searches with INSTRING(...,1,startIndex), using uppercase copies when caseSensitive is false; default start is 1.

Parameters

Parameter	Data type	Description
svalue	STRING	Text to find or compare.

startIndex	ULONG	1-based starting position.

Returns

LONG — 1-based position of the first match, or 0 if not found.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — IndexOf(STRING svalue),LONG

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('red blue red')
Number = Text.IndexOf('red')
```

Expected result

Returned value: 1.
Stored string: "red blue red" (unchanged).

Example 2 — IndexOf(STRING svalue,ULONG startIndex),LONG

```
Text.Str('red blue red')
Number = Text.IndexOf('red', 2)
```

Expected result

Returned value: 10.
Stored string: "red blue red" (unchanged).

6.6. IndexOfReg

```
IndexOfReg PROCEDURE(STRING regExpValue),LONG
IndexOfReg PROCEDURE(STRING regExpValue, ULONG startIndex),LONG
IndexOfReg PROCEDURE(STRING regExpValue, ULONG startIndex,BYTE IgnoreCase),LONG
```

Description

Calls STRPOS on main storage or a slice from startIndex, then adjusts a positive offset to absolute position.

Parameters

Parameter	Data type	Description
regExpValue	STRING	Existing caller-supplied SystemStringClass output object; receives a copy on success.
startIndex	ULONG	1-based starting position.
IgnoreCase	BYTE	TRUE selects case-insensitive comparison; FALSE selects case-sensitive comparison.

Returns

LONG — 1-based match position, or 0 if no match.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — IndexOfReg(STRING regExpValue),LONG

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('abc123xyz')
Number = Text.IndexOfReg('[0-9]+')
```

Expected result

Expected position: 4 if the installed Clarion regular-expression runtime accepts [0-9]+. Stored string remains "abc123xyz".
Pattern syntax and runtime error behavior have not been verified with this Clarion installation.

Example 2 — IndexOfReg(STRING regExpValue,ULONG startIndex),LONG

```
Text.Str('abc123xyz')
Number = Text.IndexOfReg('[0-9]+', 2)
```

Expected result

Expected position: 4 if the installed Clarion regular-expression runtime accepts [0-9]+. Stored string remains "abc123xyz".
Pattern syntax and runtime error behavior have not been verified with this Clarion installation.

Example 3 — IndexOfReg(STRING regExpValue,ULONG startIndex,BYTE IgnoreCase),LONG

```
Text.Str('abc123xyz')
Number = Text.IndexOfReg('[0-9]+', 2, FALSE)
```

Expected result

Expected position: 4 if the installed Clarion regular-expression runtime accepts [0-9]+. Stored string remains "abc123xyz".
Pattern syntax and runtime error behavior have not been verified with this Clarion installation.

6.7. LastIndexOf

```
LastIndexOf PROCEDURE(STRING svalue),LONG
LastIndexOf PROCEDURE(STRING svalue, ULONG startIndex),LONG
```

Description

Repeated forward IndexOf calls from startIndex (default 1), advancing by one, retaining last position.

Parameters

Parameter	Data type	Description
svalue	STRING	Text to find or compare.
startIndex	ULONG	1-based starting position.

Returns

LONG — last matching position at or after startIndex, or 0.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — LastIndexOf(STRING svalue),LONG

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('red blue red')
Number = Text.LastIndexOf('red')
```

Expected result

Returned value: 10.
Stored string: "red blue red" (unchanged).

Example 2 — LastIndexOf(STRING svalue,ULONG startIndex),LONG

```
Text.Str('red blue red')
Number = Text.LastIndexOf('red', 2)
```

Expected result

Returned value: 10.
Stored string: "red blue red" (unchanged).

6.8. SetCaseSensitive

```
SetCaseSensitive PROCEDURE(BYTE value)
```

Description

Stores CHOOSE(input) in the corresponding configuration flag.

Parameters

Parameter	Data type	Description
value	BYTE	Base64 helper byte; reference parameters are outputs.

Returns

No return value.

Before you call

Load text and set this flag before a search that uses the stored case setting.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('Hello')
Text.SetCaseSensitive(FALSE)
Number = Text.IndexOf('hello')
```

Expected result

Returned position: 1. Stored string remains "Hello".

6.9. StartsWith

```
StartsWith PROCEDURE(SystemStringClass svalue),BYTE
StartsWith PROCEDURE(SystemStringClass svalue, BYTE IgnoreCase),BYTE
StartsWith PROCEDURE(STRING svalue),BYTE
StartsWith PROCEDURE(STRING svalue,BYTE IgnoreCase),BYTE
```

Description

Returns TRUE/FALSE from prefix, suffix or whole-value comparison.

Parameters

Parameter	Data type	Description
svalue	SystemStringClass	Input text, object, delimiter or configuration value as used in behavior notes.
IgnoreCase	BYTE	TRUE selects case-insensitive comparison; FALSE selects case-sensitive comparison.
svalue	STRING	Text to find or compare.

Returns

BYTE — TRUE if the prefix matches; otherwise FALSE.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically. Load Other as the comparison object.
Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — StartsWith(SystemStringClass svalue),BYTE

```
! Local Data
Text SystemStringClass
Other SystemStringClass
Number LONG

 CODE
Text.Str('Hello world')
Other.Str('Hello')
Number = Text.StartsWith(Other)
```

Expected result

Returned flag: 1 (TRUE). Stored string remains "Hello world".

Example 2 — StartsWith(SystemStringClass svalue,BYTE IgnoreCase),BYTE

```
Text.Str('Hello world')
Other.Str('Hello')
Number = Text.StartsWith(Other, FALSE)
```

Expected result

Returned flag: 1 (TRUE). Stored string remains "Hello world".

Example 3 — StartsWith(STRING svalue),BYTE

```
Text.Str('Hello world')
Number = Text.StartsWith('Hello')
```

Expected result

Returned flag: 1 (TRUE). Stored string remains "Hello world".

Example 4 — StartsWith(STRING svalue,BYTE IgnoreCase),BYTE

```
Text.Str('Hello world')
Number = Text.StartsWith('Hello', FALSE)
```

Expected result

Returned flag: 1 (TRUE). Stored string remains "Hello world".

7. Split, edit and join lines

Select a method below. Detailed source evidence is available through each method’s Technical notes link.

AddLine
CleanLines
CountLines
DeleteLine
FromLines
FromLinesWithEOL
GetEOLMarker
GetLine
GetLines
GetLinesCount
GetLineTrimValue
GetLineValue
InsertLine
Merge
SetEOLMarker
SetLineValue
Split
SplitToLines
TryGetLineValue

7.1. AddLine

```
AddLine PROCEDURE(STRING svalue)
```

Description

Allocates a new owned SystemStringClass, copies input into it and ADDs to lines.

Parameters

Parameter	Data type	Description
svalue	STRING	Input text.

Returns

No return value.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('red,green,blue')
Text.Split(',')
Text.AddLine('gold')
Number = Text.CountLines()
Result = Text.GetLineValue(4)
```

Expected result

Line count: 4; fourth line: "gold". Main string remains "red,green,blue".

7.2. CleanLines

```
CleanLines PROCEDURE()
```

Description

Disposes every referenced SystemStringClass in the queue and FREEs queue records.

Parameters

None.

Returns

No return value.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('red,green,blue')
Text.Split(',')
Text.CleanLines()
Number = Text.CountLines()
```

Expected result

Line count is 0. Stored string remains "red,green,blue".

7.3. CountLines

```
CountLines PROCEDURE(),ULONG
```

Description

Returns RECORDS(lines); no automatic split or synchronization with main content.

Parameters

None.

Returns

ULONG — count or current position.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example

```
! Local Data
```

```
Text SystemStringClass
Number LONG

 CODE
Text.Str('red,green,blue')
Text.Split(',')
Number = Text.CountLines()
```

Expected result

Returned count: 3. Without Split/AddLine the new object has 0 line records, even if its main string contains commas.

7.4. DeleteLine

```
DeleteLine PROCEDURE(ULONG lineNumber)
```

Description

GETs line and, on successful GET, disposes child and DELETEs record.

Parameters

Parameter	Data type	Description
lineNumber	ULONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.

Returns

No return value.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('red,green,blue')
Text.Split(',')
Text.DeleteLine(2)
Number = Text.CountLines()
Text.Merge(',')
Result = Text.Str()
```

Expected result

Line count: 2. Main text is "red,blue" after Merge; deleting alone leaves the main text unchanged.

7.5. FromLines

```
FromLines PROCEDURE()
FromLines PROCEDURE(STRING delimiter)
FromLines PROCEDURE(STRING leftDelimiter, STRING rigthDelimiter)
FromLines PROCEDURE(STRING leftDelimiter, STRING rigthDelimiter, BYTE omitFirstSeparator, BYTE
omitLastSeparator)
```

Description

Replaces main content by concatenating line values.

Parameters

Parameter	Data type	Description
delimiter	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
leftDelimiter	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
rigthDelimiter	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
omitFirstSeparator	BYTE	Omit first left delimiter.
omitLastSeparator	BYTE	Omit last right delimiter; first-branch precedence applies to a single line.

Returns

No return value.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example 1 — FromLines()

```
! Local Data
Text SystemStringClass
Result STRING(1024)
```

```
 CODE
Text.Str('red,green,blue')
Text.Split(',')
Text.FromLines()
Result = Text.Str()
```

Expected result

Stored string and Result: "redgreenblue". Line records are retained.

Example 2 — FromLines(STRING delimiter)

```
Text.Str('red,green,blue')
Text.Split(',')
Text.FromLines('|')
Result = Text.Str()
```

Expected result

Stored string and Result: "red|green|blue|". Line records are retained.

Example 3 — FromLines(STRING leftDelimiter,STRING rigthDelimiter)

```
Text.Str('red,green,blue')
Text.Split(',')
Text.FromLines('[', ']')
Result = Text.Str()
```

Expected result

Stored string and Result: "[red][green][blue]". Line records are retained.

Example 4 — FromLines(STRING leftDelimiter,STRING rigthDelimiter,BYTE omitFirstSeparator,BYTE omitLastSeparator)

```
Text.Str('red,green,blue')
Text.Split(',')
Text.FromLines('[', ']', TRUE, TRUE)
Result = Text.Str()
```

Expected result

Stored string and Result: "red][green][blue". Line records are retained.

Notes

With one line and both omission flags set, the right delimiter is still appended.

7.6. FromLinesWithEOL

```
FromLinesWithEOL PROCEDURE()
```

Description

Replaces main content by concatenating line values.

Parameters

None.

Returns

No return value.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('red,green,blue')
Text.Split(',')
Text.SetEOLMarker('|')
Text.FromLinesWithEOL()
Result = Text.Str()
```

Expected result

Stored string and Result: "red|green|blue|". Line records are retained.

7.7. GetEOLMarker

```
GetEOLMarker PROCEDURE(),STRING
```

Description

Returns the corresponding stored delimiter by value.

Parameters

None.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

No string needs to be loaded to read or change this setting. Configure it before the operation that uses it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Result = Text.GetEOLMarker()
```

Expected result

New-object result: CR followed by LF (bytes 13,10). These are control bytes, not the literal characters "<13,10>".

7.8. GetLine

```
GetLine PROCEDURE(ULONG lineNumber),*SystemStringClass
```

Description

Uses a 1-based line number; caller must enforce >=1.

Parameters

Parameter	Data type	Description
lineNumber	ULONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.

Returns

Borrowed SystemStringClass reference, or NULL when lineNumber exceeds the count.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
```

```
LineRef &SystemStringClass

 CODE
Text.Str('red,green,blue')
Text.Split(',')
LineRef &= Text.GetLine(2)
IF NOT LineRef &= NULL
 Result = LineRef.Str()
END
```

Expected result

Borrowed line object contains "green". Do not dispose LineRef; resplitting or clearing the queue invalidates it.

7.9. GetLines

```
GetLines PROCEDURE(),*SystemStringQueue
```

Description

Returns borrowed mutable lines queue.

Parameters

None.

Returns

Borrowed reference to the mutable line queue.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example

```
! Local Data
Text SystemStringClass
Number LONG
QueueRef &SystemStringQueue

 CODE
Text.Str('red,green,blue')
Text.Split(',')
QueueRef &= Text.GetLines()
Number = RECORDS(QueueRef)
```

Expected result

Returned reference points to the owned line queue; record count is 3. Do not dispose the queue or its children.

7.10. GetLinesCount

```
GetLinesCount PROCEDURE(),ULONG
```

Description

Returns RECORDS(lines); no automatic split or synchronization with main content.

Parameters

None.

Returns

ULONG — count or current position.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('red,green,blue')
Text.Split(',')
Number = Text.GetLinesCount()
```

Expected result

Returned count: 3. Without Split/AddLine the new object has 0 line records, even if its main string contains commas.

7.11. GetLineTrimValue

```
GetLineTrimValue PROCEDURE(ULONG lineNumber),STRING
```

Description

Uses a 1-based line number; caller must enforce >=1.

Parameters

Parameter	Data type	Description
lineNumber	ULONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str(' red , blue ')
Text.Split(',')
Result = Text.GetLineTrimValue(1)
```

Expected result

Returned text: "red". Stored first line remains " red "; main string remains " red , blue ".

7.12. GetLineValue

```
GetLineValue PROCEDURE(ULONG lineNumber),STRING
```

Description

Uses a 1-based line number; caller must enforce >=1.

Parameters

Parameter	Data type	Description
lineNumber	ULONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('red,green,blue')
Text.Split(',')
Result = Text.GetLineValue(2)
```

Expected result

Returned text: "green". Main string unchanged.

7.13. InsertLine

```
InsertLine PROCEDURE(ULONG startIndex, STRING svalue)
```

Description

Allocates a new owned SystemStringClass, copies input into it and ADDs to lines.

Parameters

Parameter	Data type	Description
startIndex	ULONG	1-based starting position.
svalue	STRING	Input text.

Returns

No return value.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
```

```
 CODE
Text.Str('red,green,blue')
Text.Split(',')
Text.InsertLine(1, 'gold')
Result = Text.GetLineValue(1)
```

Expected result

First line: "gold". Main string remains "red,green,blue". Queue insertion position follows Clarion ADD semantics.

7.14. Merge

```
Merge PROCEDURE(STRING itemDelimiter)
Merge PROCEDURE()
```

Description

Replaces main content by concatenating line values.

Parameters

Parameter	Data type	Description
itemDelimiter	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.
Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it. Merge() reuses the last Split delimiter.

Example 1 — Merge(STRING itemDelimiter)

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('red,green,blue')
Text.Split(',')
Text.Merge('|')
Result = Text.Str()
```

Expected result

Stored string and Result: "red|green|blue". Line records are retained.

Example 2 — Merge()

```
Text.Str('red,green,blue')
Text.Split(',')
Text.Merge()
Result = Text.Str()
```

Expected result

Stored string and Result: "red,green,blue". Line records are retained.

Notes

A one-line merge retains the right delimiter because the first-line branch takes precedence.

7.15. SetEOLMarker

```
SetEOLMarker PROCEDURE(STRING EOLMarker)
```

Description

Assigns delimiter configuration.

Parameters

Parameter	Data type	Description
EOLMarker	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.

Before you call

Set the marker before AppendLine, Split() or FromLinesWithEOL uses it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.SetEOLMarker('|')
Text.Str('cat')
```

```
Text.AppendLine('dog')
Result = Text.Str()
```

Expected result

Stored string and Result: "catdog|". AppendLine adds the marker after its new input; it does not insert one before dog.

7.16. SetLineValue

```
SetLineValue PROCEDURE(ULONG lineNumber, STRING lineVal),BYTE,PROC
```

Description

Uses a 1-based line number; caller must enforce >=1.

Parameters

Parameter	Data type	Description
lineNumber	ULONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.
lineVal	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

BYTE — TRUE when the line was updated; otherwise FALSE.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('red,green,blue')
Text.Split(',')
Number = Text.SetLineValue(2, 'gold')
Result = Text.GetLineValue(2)
Text.Merge(',')
```

Expected result

Returned flag: 1; second line becomes "gold". Only after Merge does main text become "red,gold,blue".

7.17. Split

```
Split PROCEDURE()
Split PROCEDURE(STRING lineSeparator,BYTE includeEmptyResults = false)
```

Description

Disposes previous line objects and creates copied line objects, leaving main content unchanged.

Parameters

Parameter	Data type	Description
lineSeparator	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
false	BYTE includeEmptyResults =	Base64 helper byte; reference parameters are outputs.

Returns

No return value.

Before you call

Load EOL-delimited text; Split() uses the configured marker and copies lines into a separate queue.
Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example 1 — Split()

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('red<13,10>blue')
Text.Split()
Number = Text.CountLines()
Result = Text.GetLineValue(2)
```

Expected result

Line count: 2; second line: "blue". Main string remains red, CR/LF, blue.

Example 2 — Split(STRING lineSeparator,BYTE includeEmptyResults = false)

```
Text.Str('red,green,blue')
Text.Split(',')
Number = Text.CountLines()
Result = Text.GetLineValue(2)
```

Expected result

Line count: 3; second line: "green". Main string remains "red,green,blue".

Notes

A trailing separator normally does not produce a trailing empty line. Avoid an empty separator.

7.18. SplitToLines

```
SplitToLines PROCEDURE(LONG lineLength=0)
```

Description

Disposes previous line objects and creates copied line objects, leaving main content unchanged.

Parameters

Parameter	Data type	Description
lineLength=0	LONG	Requested byte length/count or width. See allocation and range restrictions.

Returns

No return value.

Before you call

Load the text. A positive length creates fixed-width line records; zero instead uses Split() and the EOL marker.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('abcdefg')
Text.SplitToLines(3)
Number = Text.CountLines()
Result = Text.GetLineValue(3)
```

Expected result

Lines are "abc", "def", "g"; count is 3. Returned third line: "g". Main string remains "abcdefg".

Notes

Use zero or a positive line length; negative lengths are not validated.

7.19. TryGetLineValue

```
TryGetLineValue PROCEDURE(ULONG lineNumber,*SystemStringClass pValue),BYTE
```

Description

Uses a 1-based line number; caller must enforce >=1.

Parameters

Parameter	Data type	Description
lineNumber	ULONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.
pValue	*SystemStringClass	Existing caller-supplied SystemStringClass output object; receives a copy on success.

Returns

BYTE — TRUE when a line was copied to pValue; otherwise FALSE.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example

```
! Local Data
Text SystemStringClass
Other SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('red,green,blue')
Text.Split(',')
Number = Text.TryGetLineValue(2, Other)
Result = Other.Str()
```

Expected result

Returned flag: 1. Other receives a copy of "green". Main string unchanged.

8. Tokens and words

Select a method below. Detailed source evidence is available through each method's Technical notes link.

CountToken
CountTokens
CountWord
CountWords
FoundToken
GetCurrentToken
GetToken
GetTokensDelimiters
GetUseLowerAsciiAsDelimiters
GetWord
IsPastEOL
IsTokenEmpty
IsTokenInited
IsTokenNull
NextToken
PopToken
PushToken
ResetToken
SetTokensDelimiters
SetUseLowerAsciiAsDelimiters

8.1. CountToken

```
CountToken PROCEDURE(STRING svalue),LONG
```

Description

Count/Get methods reset and advance token selection; not state-preserving queries.

Parameters

Parameter	Data type	Description
svalue	STRING	Input text.

Returns

LONG — count or current position.

Before you call

Load text and set the token delimiters. ResetToken starts traversal; NextToken selects the first token. Selecting/counting tokens changes traversal state, not the stored text.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('red,blue,red')
Text.SetTokensDelimiters(',')
Number = Text.CountToken('red')
```

Expected result

Returned count: 2. Stored string remains "red,blue,red". CountToken performs its own reset/traversal.

8.2. CountTokens

```
CountTokens PROCEDURE(),LONG
```

Description

Count/Get methods reset and advance token selection; not state-preserving queries.

Parameters

None.

Returns

LONG — count or current position.

Before you call

Load text and set the token delimiters. ResetToken starts traversal; NextToken selects the first token. Selecting/counting tokens changes traversal state, not the stored text.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('red,green,blue')
Text.SetTokensDelimiters(',')
Text.ResetToken()
Number = Text.CountTokens()
```

Expected result

Returned count: 3. CountTokens resets and consumes traversal; reset again before starting another scan. Stored string: "red,green,blue" (unchanged).

8.3. CountWord

```
CountWord PROCEDURE(STRING svalue),LONG
```

Description

Count/Get methods reset and advance token selection; not state-preserving queries.

Parameters

Parameter	Data type	Description
svalue	STRING	Input text.

Returns

LONG — count or current position.

Before you call

Load space-separated text and explicitly set space as the delimiter. These methods handle their own traversal.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('red blue red')
Text.SetTokensDelimiters(' ')
Number = Text.CountWord('red')
```

Expected result

Returned count: 2. Stored string remains "red blue red".

8.4. CountWords

```
CountWords PROCEDURE(),LONG
```

Description

Count/Get methods reset and advance token selection; not state-preserving queries.

Parameters

None.

Returns

LONG — count or current position.

Before you call

Load space-separated text and explicitly set space as the delimiter. These methods handle their own traversal.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('red blue red')
Text.SetTokensDelimiters(' ')
Number = Text.CountWords()
```

Expected result

Returned count: 3. Stored string remains "red blue red".

8.5. FoundToken

```
FoundToken PROCEDURE(STRING svalue),LONG
FoundToken PROCEDURE(STRING svalue, ULONG startIndex),LONG
```

Description

Finds only the first IndexOf candidate at/after startIndex, then checks boundaries; does not retry later candidates.

Parameters

Parameter	Data type	Description
svalue	STRING	Text to find or compare.
startIndex	ULONG	1-based starting position.

Returns

LONG — matching token position, or 0; also changes token selection.

Before you call

Load text and set delimiters. FoundToken also changes the selected token. Use a single-byte token here to avoid its documented multi-byte boundary defect.

Example 1 — FoundToken(STRING svalue),LONG

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('A,B,C')
Text.SetTokensDelimiters(',')
Number = Text.FoundToken('B')
Result = Text.GetCurrentToken()
```

Expected result

Returned position: 3. Selected token: "B". Stored string remains "A,B,C".

Example 2 — FoundToken(STRING svalue,ULONG startIndex),LONG

```
Text.Str('A,B,C')
Text.SetTokensDelimiters(',')
Number = Text.FoundToken('B', 2)
Result = Text.GetCurrentToken()
```

Expected result

Returned position: 3. Selected token: "B". Stored string remains "A,B,C".

Notes

The right-boundary check uses start+1 rather than end+1, and later matches are not retried.

8.6. GetCurrentToken

```
GetCurrentToken PROCEDURE(),STRING
```

Description

Returns empty if selection is null or a single delimiter.

Parameters

None.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load text and set the token delimiters. ResetToken starts traversal; NextToken selects the first token. Selecting/counting tokens changes traversal state, not the stored text.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('red,green,blue')
Text.SetTokensDelimiters(',')
Text.ResetToken()
Result = Text.NextToken()
Result = Text.GetCurrentToken()
```

Expected result

Returned text: "red". GetCurrentToken does not advance to green.
Stored string: "red,green,blue" (unchanged).

8.7. GetToken

```
GetToken PROCEDURE(ULONG tokenIndex),STRING
```

Description

Count/Get methods reset and advance token selection; not state-preserving queries.

Parameters

Parameter	Data type	Description
tokenIndex	ULONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load text and set the token delimiters. ResetToken starts traversal; NextToken selects the first token. Selecting/counting tokens changes traversal state, not the stored text.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('red,green,blue')
Text.SetTokensDelimiters(',')
```

```
Text.ResetToken()
Result = Text.GetToken(2)
```

Expected result

Returned text: "green". GetToken performs its own reset/traversal; no prior NextToken is required.
Stored string: "red,green,blue" (unchanged).

8.8. GetTokensDelimiters

```
GetTokensDelimiters PROCEDURE(),STRING
```

Description

Returns the corresponding stored delimiter by value.

Parameters

None.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

No string needs to be loaded to read or change this setting. Configure it before the operation that uses it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.SetTokensDelimiters(',;')
Result = Text.GetTokensDelimiters()
```

Expected result

Result: ",;". The example sets a short explicit list instead of reproducing the full default list.

8.9. GetUseLowerAsciiAsDelimiters

```
GetUseLowerAsciiAsDelimiters PROCEDURE(),BYTE
```

Description

Returns the corresponding stored BYTE configuration flag.

Parameters

None.

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

No string needs to be loaded to read or change this setting. Configure it before the operation that uses it.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Number = Text.GetUseLowerAsciiAsDelimiters()
```

Expected result

New-object result: 0 (FALSE).

8.10. GetWord

```
GetWord PROCEDURE(ULONG wordIndex),STRING
```

Description

Count/Get methods reset and advance token selection; not state-preserving queries.

Parameters

Parameter	Data type	Description
wordIndex	ULONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load space-separated text and explicitly set space as the delimiter. These methods handle their own traversal.

Example

```
! Local Data
```

```
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('red blue red')
Text.SetTokensDelimiters(' ')
Result = Text.GetWord(2)
```

Expected result

Returned text: "blue". Stored string remains "red blue red".

8.11. IsPastEOL

```
IsPastEOL PROCEDURE(),BYTE
```

Description

TRUE for NULL storage, reversed selection, or start beyond length except the special length+1/length+1 state.

Parameters

None.

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

Load text and set the token delimiters. ResetToken starts traversal; NextToken selects the first token. Selecting/counting tokens changes traversal state, not the stored text.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('red,green,blue')
Text.SetTokensDelimiters(',')
Text.ResetToken()
Result = Text.NextToken()
Number = Text.IsPastEOL()
```

Expected result

Selected token: "red". Returned flag: 0 (FALSE): traversal has not passed the stored text.
Stored string: "red,green,blue" (unchanged).

8.12. IsTokenEmpty

```
IsTokenEmpty PROCEDURE(),BYTE
```

Description

TRUE for NULL storage or selection on one delimiter byte.

Parameters

None.

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

Load text and set the token delimiters. ResetToken starts traversal; NextToken selects the first token. Selecting/counting tokens changes traversal state, not the stored text.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('red,green,blue')
Text.SetTokensDelimiters(',')
Text.ResetToken()
Result = Text.NextToken()
Number = Text.IsTokenEmpty()
```

Expected result

Selected token: "red". Returned flag: 0 (FALSE). Consecutive delimiters can instead select an empty token. Stored string: "red,green,blue" (unchanged).

8.13. IsTokenInited

```
IsTokenInited PROCEDURE(),BYTE
```

Description

Returns FALSE when both selection fields are zero, TRUE otherwise.

Parameters

None.

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

Load text and set the token delimiters. ResetToken starts traversal; NextToken selects the first token. Selecting/counting tokens changes traversal state, not the stored text.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('red,green,blue')
Text.SetTokensDelimiters(',')
Text.ResetToken()
Result = Text.NextToken()
Number = Text.IsTokenInited()
```

Expected result

Selected token: "red". Returned flag: 1 (TRUE).
Stored string: "red,green,blue" (unchanged).

Notes

Returns TRUE after traversal starts, contrary to the declaration comment.

8.14. IsTokenNull

```
IsTokenNull PROCEDURE(),BYTE
```

Description

TRUE for NULL storage, reversed selection, indices beyond length or indices <1.

Parameters

None.

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

Load text and set the token delimiters. ResetToken starts traversal; NextToken selects the first token. Selecting/counting tokens changes traversal state, not the stored text.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('red,green,blue')
Text.SetTokensDelimiters(',')
Text.ResetToken()
Result = Text.NextToken()
Number = Text.IsTokenNull()
```

Expected result

Selected token: "red". Returned flag: 0 (FALSE).
Stored string: "red,green,blue" (unchanged).

8.15. NextToken

```
NextToken PROCEDURE(),STRING,PROC
```

Description

Advances selection from previous end+1, scans until delimiter or end and returns GetCurrentToken().

Parameters

None.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load text and set the token delimiters. ResetToken starts traversal; NextToken selects the first token.
Selecting/counting tokens changes traversal state, not the stored text.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('red,green,blue')
Text.SetTokensDelimiters(',')
Text.ResetToken()
Result = Text.NextToken()
Result = Text.NextToken()
```

Expected result

First return: "red"; second return: "green".
Stored string: "red,green,blue" (unchanged).

8.16. PopToken

```
PopToken PROCEDURE(),BYTE,PROC
```

Description

Push inserts value via Insert then FromString and sets selection end=startIndex+LEN(value), which may include following byte.

Parameters

None.

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

Load text and set the token delimiters. ResetToken starts traversal; NextToken selects the first token. Selecting/counting tokens changes traversal state, not the stored text.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('red,green,blue')
Text.SetTokensDelimiters(',')
Text.ResetToken()
Result = Text.NextToken()
Number = Text.PopToken()
Result = Text.Str()
```

Expected result

Returned flag: 1. Stored string becomes "green,blue": the selected red token and its following comma are removed.

8.17. PushToken

```
PushToken PROCEDURE(ULONG startIndex, STRING svalue)
```

Description

Push inserts value via Insert then FromString and sets selection end=startIndex+LEN(value), which may include following byte.

Parameters

Parameter	Data type	Description
startIndex	ULONG	1-based starting position.
svalue	STRING	Input text.

Returns

No return value.

Before you call

Load text before insertion. PushToken inserts text and also establishes a token selection; its selection-end arithmetic can include a following byte.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('green,blue')
Text.SetTokensDelimiters(',')
Text.PushToken(1, 'red,')
Result = Text.Str()
```

Expected result

Stored string: "red,green,blue". No value is returned by PushToken. This example reads the stored string rather than relying on the questionable selection end.

8.18. ResetToken

```
ResetToken PROCEDURE()
```

Description

Sets TokenSelStart and TokenSelEnd to zero.

Parameters

None.

Returns

No return value.

Before you call

Load text and set the token delimiters. ResetToken starts traversal; NextToken selects the first token. Selecting/counting tokens changes traversal state, not the stored text.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('red,green,blue')
Text.SetTokensDelimiters(',')
Text.ResetToken()
Result = Text.NextToken()
Text.ResetToken()
Number = Text.IsTokenInited()
```

Expected result

NextToken initially returns "red"; after ResetToken, IsTokenInited returns 0.
Stored string: "red,green,blue" (unchanged).

8.19. SetTokensDelimiters

```
SetTokensDelimiters PROCEDURE(STRING delimiters)
```

Description

Assigns delimiter configuration.

Parameters

Parameter	Data type	Description
delimiters	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.

Before you call

Load text and configure delimiters before token traversal. A delimiter list is a set of individual bytes.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('red;green;blue')
Text.SetTokensDelimiters(';')
Result = Text.GetToken(2)
```

Expected result

Returned token: "green". Main string remains "red;green;blue". GetToken performs its own traversal.

8.20. SetUseLowerAsciiAsDelimiters

```
SetUseLowerAsciiAsDelimiters PROCEDURE(BYTE value=true)
```

Description

Stores CHOOSE(input) in the corresponding configuration flag.

Parameters

Parameter	Data type	Description
value=true	BYTE	Base64 helper byte; reference parameters are outputs.

Returns

No return value.

Before you call

Use this setting before token traversal when control bytes should separate tokens.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('red<9>green')
Text.SetTokensDelimiters(',')
Text.SetUseLowerAsciiAsDelimiters(TRUE)
Number = Text.CountTokens()
```

Expected result

Returned count: 2. Stored bytes are red, TAB (9), green. Main text is unchanged.
Control-byte token extraction has edge cases; this count uses the supplied source algorithm, not a runtime test.

9. Change case, trim and pad

Select a method below. Detailed source evidence is available through each method's Technical notes link.
PadCenter
PadLeft
PadRight
ToCapitalize
ToCapitalizeInContext
ToLower
ToLowerInContext
ToUpper
ToUpperInContext
Trim
TrimEnd
TrimInContext
TrimStart

9.1. PadCenter

```
PadCenter PROCEDURE(ULONG totalWidth),STRING
PadCenter PROCEDURE(ULONG totalWidth, STRING paddingChar),STRING
```

Description

Returns padded value without mutation.

Parameters

Parameter	Data type	Description
totalWidth	ULONG	Requested byte length/count or width. See allocation and range restrictions.
paddingChar	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — PadCenter(ULONG totalWidth),STRING

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('cat')
Result = Text.PadCenter(6)
```

Expected result

Returned text: " cat ".
Stored string: "cat" (unchanged).

Example 2 — PadCenter(ULONG totalWidth,STRING paddingChar),STRING

```
Text.Str('cat')
Result = Text.PadCenter(6, '.')
```

Expected result

Returned text: ".cat..".
Stored string: "cat" (unchanged).

9.2. PadLeft

```
PadLeft PROCEDURE(ULONG totalWidth),STRING
PadLeft PROCEDURE(ULONG totalWidth, STRING paddingChar),STRING
```

Description

Returns padded value without mutation.

Parameters

Parameter	Data type	Description
totalWidth	ULONG	Requested byte length/count or width. See allocation and range restrictions.
paddingChar	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — PadLeft(ULONG totalWidth),STRING

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('cat')
Result = Text.PadLeft(6)
```

Expected result

Returned text: " cat".
Stored string: "cat" (unchanged).

Example 2 — PadLeft(ULONG totalWidth,STRING paddingChar),STRING

```
Text.Str('cat')
Result = Text.PadLeft(6, '.')
```

Expected result

Returned text: "...cat".
Stored string: "cat" (unchanged).

9.3. PadRight

```
PadRight PROCEDURE(ULONG totalWidth),STRING
PadRight PROCEDURE(ULONG totalWidth, STRING paddingChar),STRING
```

Description

Returns padded value without mutation.

Parameters

Parameter	Data type	Description
totalWidth	ULONG	Requested byte length/count or width. See allocation and range restrictions.
paddingChar	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — PadRight(ULONG totalWidth),STRING

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('cat')
Result = Text.PadRight(6)
```

Expected result

Returned text: "cat ".
Stored string: "cat" (unchanged).

Example 2 — PadRight(ULONG totalWidth,STRING paddingChar),STRING

```
Text.Str('cat')
Result = Text.PadRight(6, '.')
```

Expected result

Returned text: "cat...".
Stored string: "cat" (unchanged).

9.4. ToCapitalize

```
ToCapitalize PROCEDURE(),STRING
ToCapitalize PROCEDURE(STRING pStr),STRING
```

Description

Uppercases first character and characters immediately after literal spaces; leaves other letters unchanged.

Parameters

Parameter	Data type	Description
pStr	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — ToCapitalize(),STRING

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('hello clarion world')
Result = Text.ToCapitalize()
```

Expected result

Returned text: "Hello Clarion World".
Stored string: "hello clarion world" (unchanged).

Example 2 — ToCapitalize(STRING pStr),STRING

```
Text.Str('hello clarion world')
Result = Text.ToCapitalize('hello clarion world')
```

Expected result

Returned text: "Hello Clarion World".
Stored string: "hello clarion world" (unchanged).

9.5. ToCapitalizeInContext

```
ToCapitalizeInContext PROCEDURE()
```

Description

Uppercases first character and characters immediately after literal spaces; leaves other letters unchanged.

Parameters

None.

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('hello clarion world')
Text.ToCapitalizeInContext()
Result = Text.Str()
```

Expected result

Stored string: "Hello Clarion World". This call has no return value; Result reads the stored string afterward.

9.6. ToLower

```
ToLower PROCEDURE(),STRING
ToLower PROCEDURE(STRING pStr),STRING
```

Description

Returns LOWER/UPPER(SELF.Str()) without content mutation.

Parameters

Parameter	Data type	Description
pStr	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — ToLower(),STRING

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('Hello WORLD')
Result = Text.ToLower()
```

Expected result

Returned text: "hello world".
Stored string: "Hello WORLD" (unchanged).

Example 2 — ToLower(STRING pStr),STRING

```
Text.Str('Hello WORLD')
Result = Text.ToLower('different text')
```

Expected result

Returned text: "hello world".
Stored string: "Hello WORLD" (unchanged).
The supplied pStr is ignored by this implementation. The result comes from Text.Str(), despite the argument.

Notes

The pStr overload ignores its argument and converts the current object value.

9.7. ToLowerInContext

```
ToLowerInContext PROCEDURE()
```

Description

Replaces main value with LOWER/UPPER(Str()), which can include cursor clipping.

Parameters

None.

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('Hello WORLD')
Text.ToLowerInContext()
Result = Text.Str()
```

Expected result

Stored string: "hello world". This call has no return value; Result reads the stored string afterward.

9.8. ToUpper

```
ToUpper PROCEDURE(),STRING
ToUpper PROCEDURE(STRING pStr),STRING
```

Description

Returns LOWER/UPPER(SELF.Str()) without content mutation.

Parameters

Parameter	Data type	Description
pStr	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — ToUpper(),STRING

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('Hello World')
Result = Text.ToUpper()
```

Expected result

Returned text: "HELLO WORLD".
Stored string: "Hello World" (unchanged).

Example 2 — ToUpper(STRING pStr),STRING

```
Text.Str('Hello World')
Result = Text.ToUpper('different text')
```

Expected result

Returned text: "HELLO WORLD".
Stored string: "Hello World" (unchanged).
The supplied pStr is ignored by this implementation. The result comes from Text.Str(), despite the argument.

Notes

The pStr overload ignores its argument and converts the current object value.

9.9. ToUpperInContext

```
ToUpperInContext PROCEDURE()
```

Description

Replaces main value with LOWER/UPPER(Str()), which can include cursor clipping.

Parameters

None.

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('Hello World')
Text.ToUpperInContext()
Result = Text.Str()
```

Expected result

Stored string: "HELLO WORLD". This call has no return value; Result reads the stored string afterward.

9.10. Trim

```
Trim PROCEDURE(),STRING
Trim PROCEDURE(STRING char1,<STRING char2>,<STRING char3>,<STRING char4>),STRING
Trim PROCEDURE(BYTE trimType,STRING char1,<STRING char2>,<STRING char3>,<STRING char4>),STRING
```

Description

trimType: 0 both, 1 left, 2 right.

Parameters

Parameter	Data type	Description
char1	STRING	Trim candidate; only first byte is used.
char2	STRING	Trim candidate; only first byte is used. Optional (omittable).
char3	STRING	Trim candidate; only first byte is used. Optional (omittable).
char4	STRING	Trim candidate; only first byte is used. Optional (omittable).
trimType	BYTE	0 both ends; 1 left; 2 right.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — Trim(),STRING

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str(' cat ')
Result = Text.Trim()
```

Expected result

Returned text: "cat".
Stored string: " cat " (unchanged).

Example 2 — Trim(STRING char1,<STRING char2>,<STRING char3>,<STRING char4>),STRING

```
Text.Str('--cat--')
Result = Text.Trim('-')
```

Expected result

Returned text: "cat".
Stored string: "--cat--" (unchanged).

Example 3 — Trim(BYTE trimType,STRING char1,<STRING char2>,<STRING char3>,<STRING char4>),STRING

```
Text.Str('--cat--')
Result = Text.Trim(0, '-')
```

Expected result

Returned text: "cat".
Stored string: "--cat--" (unchanged).

9.11. TrimEnd

```
TrimEnd PROCEDURE(STRING char1,<STRING char2>,<STRING char3>,<STRING char4>),STRING
```

Description

trimType: 0 both, 1 left, 2 right.

Parameters

Parameter	Data type	Description
char1	STRING	Trim candidate; only first byte is used.
char2	STRING	Trim candidate; only first byte is used. Optional (omittable).
char3	STRING	Trim candidate; only first byte is used. Optional (omittable).
char4	STRING	Trim candidate; only first byte is used. Optional (omittable).

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('--cat--')
Result = Text.TrimEnd('-')
```

Expected result

Returned text: "--cat".
Stored string: "--cat--" (unchanged).

9.12. TrimInContext

```
TrimInContext PROCEDURE()
TrimInContext PROCEDURE(BYTE trimType,STRING char1,<STRING char2>,<STRING char3>,<STRING char4>)
TrimInContext PROCEDURE(STRING char1,<STRING char2>,<STRING char3>,<STRING char4>)
```

Description

trimType: 0 both, 1 left, 2 right.

Parameters

Parameter	Data type	Description
trimType	BYTE	0 both ends; 1 left; 2 right.
char1	STRING	Trim candidate; only first byte is used.
char2	STRING	Trim candidate; only first byte is used. Optional (omittable).
char3	STRING	Trim candidate; only first byte is used. Optional (omittable).
char4	STRING	Trim candidate; only first byte is used. Optional (omittable).

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — TrimInContext()

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str(' cat ')
Text.TrimInContext()
Result = Text.Str()
```

Expected result

Stored string: "cat". This call has no return value; Result reads the stored string afterward.

Example 2 — TrimInContext(BYTE trimType,STRING char1,<STRING char2>,<STRING char3>,<STRING char4>)

```
Text.Str('--cat--')
Text.TrimInContext(0, '-')
Result = Text.Str()
```

Expected result

Stored string: "cat". This call has no return value; Result reads the stored string afterward.

Example 3 — TrimInContext(STRING char1,<STRING char2>,<STRING char3>,<STRING char4>)

```
Text.Str('--cat--')
Text.TrimInContext('-')
Result = Text.Str()
```

Expected result

Stored string: "cat". This call has no return value; Result reads the stored string afterward.

Notes

Right-only trimming can be skipped by the helper’s return condition.

9.13. TrimStart

```
TrimStart PROCEDURE(STRING char1,<STRING char2>,<STRING char3>,<STRING char4>),STRING
```

Description

trimType: 0 both, 1 left, 2 right.

Parameters

Parameter	Data type	Description
char1	STRING	Trim candidate; only first byte is used.
char2	STRING	Trim candidate; only first byte is used. Optional (omittable).
char3	STRING	Trim candidate; only first byte is used. Optional (omittable).
char4	STRING	Trim candidate; only first byte is used. Optional (omittable).

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('--cat--')
Result = Text.TrimStart('-')
```

Expected result

Returned text: "cat--".
Stored string: "--cat--" (unchanged).

10. Format and process key/value pairs

Select a method below. Detailed source evidence is available through each method's Technical notes link.
ExtractKeyValue
FormatString
GetPairSeparator
GetValueSeparator
ReplaceKeyValue
SetPairSeparator
SetPairValueSeparator
SetValueSeparator

10.1. ExtractKeyValue

```
ExtractKeyValue PROCEDURE(STRING pKey, STRING pValueSep, STRING pPairSep, *SystemStringClass pValue),BYTE
ExtractKeyValue PROCEDURE(STRING pKey, *SystemStringClass pValue),BYTE
```

Description

Searches CLIP(key)&CLIP(value separator) anywhere, without enforcing pair boundary.

Parameters

Parameter	Data type	Description
pKey	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
pValueSep	STRING	Existing caller-supplied SystemStringClass output object; receives a copy on success.
pPairSep	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
pValue	*SystemStringClass	Existing caller-supplied SystemStringClass output object; receives a copy on success.

Returns

BYTE — TRUE when found and copied to pValue; FALSE leaves pValue unchanged.

Before you call

Load key/value text. Other must already be a valid SystemStringClass object to receive the copied value. The short overload uses stored separators (= and & by default).

Example 1 — ExtractKeyValue(STRING pKey,STRING pValueSep,STRING pPairSep,*SystemStringClass pValue),BYTE

```
! Local Data
Text SystemStringClass
Other SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('name=cat&age=3')
Number = Text.ExtractKeyValue('name', '=', '&', Other)
Result = Other.Str()
```

Expected result

Returned flag: 1. Other receives "cat"; main string remains "name=cat&age=3".

Example 2 — ExtractKeyValue(STRING pKey,*SystemStringClass pValue),BYTE

```
Text.Str('name=cat&age=3')
Number = Text.ExtractKeyValue('name', Other)
Result = Other.Str()
```

Expected result

Returned flag: 1. Other receives "cat"; main string remains "name=cat&age=3".

10.2. FormatString

```
FormatString PROCEDURE(STRING objectsString, <STRING objectsString1>, <STRING objectsString2>, <STRING
objectsString3>, <STRING objectsString4>, <STRING objectsString5>),STRING
```

Description

Returns a formatted value, does not assign main content.

Parameters

Parameter	Data type	Description
objectsString	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
objectsString1	STRING	Input text, object, delimiter or configuration value as used in behavior notes. Optional (omittable).
objectsString2	STRING	Input text, object, delimiter or configuration value as used in behavior notes. Optional (omittable).
objectsString3	STRING	Input text, object, delimiter or configuration value as used in behavior notes. Optional (omittable).
objectsString4	STRING	Input text, object, delimiter or configuration value as used in behavior notes. Optional (omittable).
objectsString5	STRING	Input text, object, delimiter or configuration value as used in behavior notes. Optional (omittable).

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load a format string containing numbered placeholders. Clarion string literals require a doubled opening brace to represent one literal opening brace.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('Hello {{0}, {{1}!')
Result = Text.FormatString('Clarion', 'world')
```

Expected result

Returned text: "Hello Clarion, world!". Stored format string remains "Hello {0}, {1}!".

10.3. GetPairSeparator

```
GetPairSeparator PROCEDURE(),STRING
```

Description

Returns the corresponding stored delimiter by value.

Parameters

None.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

No string needs to be loaded to read or change this setting. Configure it before the operation that uses it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Result = Text.GetPairSeparator()
```

Expected result

New-object result: "&".

10.4. GetValueSeparator

```
GetValueSeparator PROCEDURE(),STRING
```

Description

Returns the corresponding stored delimiter by value.

Parameters

None.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

No string needs to be loaded to read or change this setting. Configure it before the operation that uses it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Result = Text.GetValueSeparator()
```

Expected result

New-object result: "=".

10.5. ReplaceKeyValue

```
ReplaceKeyValue PROCEDURE(STRING pKey, STRING pValueSep, STRING pPairSep, STRING pNewValue),BYTE,PROC
ReplaceKeyValue PROCEDURE(STRING pKey, STRING pNewValue),BYTE,PROC
```

Description

Searches CLIP(key)&CLIP(value separator) anywhere, without enforcing pair boundary.

Parameters

Parameter	Data type	Description
pKey	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
pValueSep	STRING	Existing caller-supplied SystemStringClass output object; receives a copy on success.
pPairSep	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
pNewValue	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

BYTE — TRUE when the key was found; does not guarantee replacement succeeded.

Before you call

Load key/value text and choose the separators before replacement. This example uses a nonfinal pair to avoid the last-pair boundary defect.

Example 1 — ReplaceKeyValue(STRING pKey,STRING pValueSep,STRING pPairSep,STRING pNewValue),BYTE,PROC

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('name=cat&age=3')
Number = Text.ReplaceKeyValue('name', '=', '&', 'dog')
Result = Text.Str()
```

Expected result

Returned flag: 1; stored string: "name=dog&age=3". A TRUE result only reports key-found in this implementation; verify the stored value for other inputs.

Example 2 — ReplaceKeyValue(STRING pKey,STRING pNewValue),BYTE,PROC

```
Text.Str('name=cat&age=3')
Number = Text.ReplaceKeyValue('name', 'dog')
Result = Text.Str()
```

Expected result

Returned flag: 1; stored string: "name=dog&age=3". A TRUE result only reports key-found in this implementation; verify the stored value for other inputs.

Notes

TRUE means the key was found; the underlying replacement may still have failed.

10.6. SetPairSeparator

```
SetPairSeparator PROCEDURE(STRING pairSeparator)
```

Description

Assigns delimiter configuration.

Parameters

Parameter	Data type	Description
pairSeparator	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.

Before you call

No string needs to be loaded to read or change this setting. Configure it before the operation that uses it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.SetPairSeparator(';')
Result = Text.GetPairSeparator()
```

Expected result

Result is ";". Stored main text is unaffected.

10.7. SetPairValueSeparator

```
SetPairValueSeparator PROCEDURE(STRING pairSeparator,STRING valueSeparator)
```

Description

Assigns delimiter configuration.

Parameters

Parameter	Data type	Description
pairSeparator	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
valueSeparator	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.

Before you call

Set both separators before the default-separator key/value overloads.

Example

```
! Local Data
Text SystemStringClass
Other SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.SetPairValueSeparator(';', ':')
Text.Str('name:cat;age:3')
Number = Text.ExtractKeyValue('name', Other)
Result = Other.Str()
```

Expected result

Returned flag: 1. Output object Other contains "cat". Main text remains "name:cat;age:3".

10.8. SetValueSeparator

```
SetValueSeparator PROCEDURE(STRING valueSeparator)
```

Description

Assigns delimiter configuration.

Parameters

Parameter	Data type	Description
valueSeparator	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.

Before you call

No string needs to be loaded to read or change this setting. Configure it before the operation that uses it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.SetValueSeparator(':')
Result = Text.GetValueSeparator()
```

Expected result

Result is ":". Stored main text is unaffected.

11. Encode, decode and compress

Select a method below. Detailed source evidence is available through each method’s Technical notes link.
Compress
DecodeBase64
Decompress
EncodeBase64
Escape
FromBase64
GetUrlEncodeDataChar
HtmlEscape

SetUrlEncodeDataChar
ToBase64
UnEscape
UrlDecode
UrlEncode

11.1. Compress

```
Compress PROCEDURE(),BYTE,PROC
```

Description

Mutates main storage only on successful runtime compression/decompression and returns TRUE, else FALSE.

Parameters

None.

Returns

BYTE — TRUE on successful compression; otherwise FALSE.

Before you call

Load text before Compress. Decompress requires the binary output of a successful Compress, not ordinary readable text.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('cat cat cat cat')
Number = Text.Compress()
IF Number
 Number = Text.Decompress()
 IF Number
 Result = Text.Str()
 END
END
```

Expected result

If both calls succeed, Result is "cat cat cat cat". Each status is TRUE on success. Compressed bytes and failure behavior depend on the installed compression runtime; no exact compressed byte sequence is asserted.

Neither compression success nor this round trip has been runtime verified for the documentation build.

11.2. DecodeBase64

```
DecodeBase64 PROCEDURE(BYTE isQuoted=0)
DecodeBase64 PROCEDURE(STRING b64sValue, BYTE isQuoted=0),STRING
```

Description

ToBase64 returns encoded copy; FromBase64 decodes input then assigns main content.

Parameters

Parameter	Data type	Description
isQuoted=0	BYTE	Enable surrounding quotes, or skip surrounding bytes when decoding.
b64sValue	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.
STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — DecodeBase64(BYTE isQuoted=0)

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('Y2F0')
Text.DecodeBase64()
Result = Text.Str()
```

Expected result

Stored string: "cat". This call has no return value; Result reads the stored string afterward.

Example 2 — DecodeBase64(STRING b64sValue,BYTE isQuoted=0),STRING

```
Text.Str('unchanged')
Result = Text.DecodeBase64('Y2F0')
```

Expected result

Returned text: "cat".
Stored string: "unchanged" (unchanged).

Notes

The decoded allocation is not shortened to the actual byte count; trailing bytes can remain.

11.3. Decompress

```
Decompress PROCEDURE(),BYTE,PROC
```

Description

Mutates main storage only on successful runtime compression/decompression and returns TRUE, else FALSE.

Parameters

None.

Returns

BYTE — TRUE on successful decompression; otherwise FALSE.

Before you call

Load text before Compress. Decompress requires the binary output of a successful Compress, not ordinary readable text.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('cat cat cat cat')
Number = Text.Compress()
IF Number
 Number = Text.Decompress()
 IF Number
 Result = Text.Str()
 END
END
```

Expected result

If both calls succeed, Result is "cat cat cat cat". Each status is TRUE on success. Compressed bytes and failure behavior depend on the installed compression runtime; no exact compressed byte sequence is asserted.
Neither compression success nor this round trip has been runtime verified for the documentation build.

11.4. EncodeBase64

```
EncodeBase64 PROCEDURE(BYTE quote=0)
EncodeBase64 PROCEDURE(STRING svalue, BYTE quote=0),STRING
```

Description

ToBase64 returns encoded copy; FromBase64 decodes input then assigns main content.

Parameters

Parameter	Data type	Description
quote=0	BYTE	Enable surrounding quotes, or skip surrounding bytes when decoding.
svalue	STRING	Input text.

Returns

No return value.
STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — EncodeBase64(BYTE quote=0)

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('cat')
Text.EncodeBase64()
Result = Text.Str()
```

Expected result

Stored string: "Y2F0". This call has no return value; Result reads the stored string afterward.

Example 2 — EncodeBase64(STRING svalue,BYTE quote=0),STRING

```
Text.Str('unchanged')
Result = Text.EncodeBase64('cat')
```

Expected result

Returned text: "Y2F0".
Stored string: "unchanged" (unchanged).

11.5. Escape

```
Escape PROCEDURE(STRING sValue,BYTE clipValue = 0),STRING
Escape PROCEDURE()
```

Description

Input overload returns transformed value and replaces owned scratch buffer, leaving main content unchanged.

Parameters

Parameter	Data type	Description
sValue	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
0	BYTE clipValue =	Existing caller-supplied SystemStringClass output object; receives a copy on success.

Returns

STRING — the resulting value; does not assign main content unless stated.
No return value.

Before you call

The text overload accepts an explicit input; loading the main text is not required. It uses object-owned scratch storage internally.
Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — Escape(STRING sValue,BYTE clipValue = 0),STRING

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('unchanged')
Result = Text.Escape('A<9>B')
```

Expected result

Returned text: "A\tB" (one backslash followed by t). Stored string remains "unchanged".

Example 2 — Escape()

```
Text.Str('A<9>B')
Text.Escape()
Result = Text.Str()
```

Expected result

Stored string: "A\tB" (literal backslash and t replacing the TAB). No value is returned by the no-argument call.

11.6. FromBase64

```
FromBase64 PROCEDURE(STRING svalue)
```

Description

ToBase64 returns encoded copy; FromBase64 decodes input then assigns main content.

Parameters

Parameter	Data type	Description
svalue	STRING	Input text.

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('old')
Text.FromBase64('Y2F0')
Result = Text.Str()
```

Expected result

Stored string: "cat". This call has no return value; Result reads the stored string afterward.

Notes

Inherits DecodeBase64 allocation and padding limitations.

11.7. GetUrlEncodeDataChar

```
GetUrlEncodeDataChar PROCEDURE(),BYTE
```

Description

Returns the corresponding stored BYTE configuration flag.

Parameters

None.

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

No string needs to be loaded to read or change this setting. Configure it before the operation that uses it.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Number = Text.GetUrlEncodeDataChar()
```

Expected result

New-object result: 0 (FALSE).

11.8. HtmlEscape

```
HtmlEscape PROCEDURE()
```

Description

Mutates main content using entities for &, <, >, apostrophe and double quote.

Parameters

None.

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('A&BZ')
Text.HtmlEscape()
Result = Text.Str()
```

Expected result

Stored string: "A&BZ". This call has no return value; Result reads the stored string afterward.

Notes

A final single-byte tail can be lost after an entity replacement.

11.9. SetUrlEncodeDataChar

```
SetUrlEncodeDataChar PROCEDURE(BYTE val)
```

Description

Stores CHOOSE(input) in the corresponding configuration flag.

Parameters

Parameter	Data type	Description
val	BYTE	Base64 helper byte; reference parameters are outputs.

Returns

No return value.

Before you call

Set this option before URL encoding. It does not itself transform the stored text.

Example

```
! Local Data
Text SystemStringClass
Number LONG
```

```
 CODE
Text.SetUrlEncodeDataChar(TRUE)
Number = Text.GetUrlEncodeDataChar()
```

Expected result

Returned flag: 1. Main text is not assigned. URL encoding remains byte-oriented and has documented limitations.

11.10. ToBase64

```
ToBase64 PROCEDURE(),STRING
```

Description

ToBase64 returns encoded copy; FromBase64 decodes input then assigns main content.

Parameters

None.

Returns

STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('cat')
Result = Text.ToBase64()
```

Expected result

Returned text: "Y2F0".
Stored string: "cat" (unchanged).

11.11. UnEscape

```
UnEscape PROCEDURE(STRING sValue,BYTE clipValue = 0),STRING
UnEscape PROCEDURE()
```

Description

Input overload returns transformed value and replaces owned scratch buffer, leaving main content unchanged.

Parameters

Parameter	Data type	Description
sValue	STRING	Input text, object, delimiter or configuration value as used in behavior notes.
0	BYTE clipValue =	Existing caller-supplied SystemStringClass output object; receives a copy on success.

Returns

STRING — the resulting value; does not assign main content unless stated.
No return value.

Before you call

The text overload accepts input directly and uses scratch storage internally.
Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — UnEscape(STRING sValue,BYTE clipValue = 0),STRING

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('unchanged')
Result = Text.UnEscape('A\tB')
```

Expected result

Returned bytes: A, TAB (9), B. Stored string remains "unchanged".

Example 2 — UnEscape()

```
Text.Str('A\tB')
Text.UnEscape()
Result = Text.Str()
```

Expected result

Stored bytes: A, TAB (9), B. No value is returned by the no-argument call.

11.12. UrlDecode

```
UrlDecode PROCEDURE()
UrlDecode PROCEDURE(STRING sValue),STRING
```

Description

Input overload returns value; no-argument overload mutates main content.

Parameters

Parameter	Data type	Description
sValue	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.
STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — UrlDecode()

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('A%20B')
Text.UrlDecode()
Result = Text.Str()
```

Expected result

Stored string: "A B". This call has no return value; Result reads the stored string afterward.

Example 2 — UrlDecode(STRING sValue),STRING

```
Text.Str('unchanged')
Result = Text.UrlDecode('A%20B')
```

Expected result

Returned text: "A B".
Stored string: "unchanged" (unchanged).

Notes

Lowercase percent escapes are decoded incorrectly; nonhex letters can be accepted.

11.13. UrlEncode

```
UrlEncode PROCEDURE()
UrlEncode PROCEDURE(STRING sValue),STRING
```

Description

Input overload returns value; no-argument overload mutates main content.

Parameters

Parameter	Data type	Description
sValue	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.
STRING — the resulting value; does not assign main content unless stated.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example 1 — UrlEncode()

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('A B')
Text.UrlEncode()
Result = Text.Str()
```

Expected result

Stored string: "A%20B". This call has no return value; Result reads the stored string afterward.

Example 2 — UrlEncode(STRING sValue),STRING

```
Text.Str('unchanged')
Result = Text.UrlEncode('A B')
```

Expected result

Returned text: "A%20B".
Stored string: "unchanged" (unchanged).

Notes

Control bytes can be discarded. This is not a standards-compliant URL codec.

12. Files and BLOBs

Select a method below. Detailed source evidence is available through each method's Technical notes link.
FromBlob
FromFile
ToBlob
ToFile

12.1. FromBlob

```
FromBlob PROCEDURE(*BLOB b)
```

Description

Copies bytes between 1-based STRING and zero-based BLOB.

Parameters

Parameter	Data type	Description
b	*BLOB	Caller-owned BLOB; input for FromBlob, output for ToBlob.

Returns

No return value.

Before you call

A valid BLOB field in an open file is required. A BLOB is not a standalone local string variable. This code assumes DemoFile.Payload is such a field and a record is available.

Example

```
! Local Data
Text SystemStringClass
Other SystemStringClass
Result STRING(1024)

 CODE
Text.Str('cat')
Text.ToBlob(DemoFile.Payload)
Other.FromBlob(DemoFile.Payload)
Result = Other.Str()
```

Expected result

On a successful BLOB copy, Payload contains the bytes "cat" and Other contains "cat". Text remains "cat". Persist the record separately using the application's normal file handling.
The field name is illustrative; define it in your own file/dictionary. BLOB I/O has not been runtime verified.

12.2. FromFile

```
FromFile PROCEDURE(STRING fileName),BYTE,PROC
```

Description

Returns BYTE-declared status: zero success, BadFileErr or ERRORCODE on failure; do not interpret as boolean success.

Parameters

Parameter	Data type	Description
fileName	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

BYTE — 0 on success; BadFileErr or a DOS driver error on failure.

Before you call

Choose a disposable, writable file path. ToFile writes it; FromFile requires a file containing the bytes to read. This example creates its own input.

Example

```
! Local Data
Text SystemStringClass
Other SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('cat')
Number = Text.ToFile('SystemString-doc-example.tmp')
IF Number = 0
 Other.Str('old')
 Number = Other.FromFile('SystemString-doc-example.tmp')
 IF Number = 0
 Result = Other.Str()
 END
END
```

Expected result

If both statuses are 0, the file bytes and Result are "cat". Text remains "cat"; Other becomes "cat" after reading. The example leaves the .tmp file in the current working directory.
File creation and driver behavior require runtime verification. Use a disposable directory; ToFile can replace existing content.

Notes

Returns zero on success. An empty file clears the value and returns BadFileErr.

12.3. ToBlob

```
ToBlob PROCEDURE(*BLOB b)
```

Description

Copies bytes between 1-based STRING and zero-based BLOB.

Parameters

Parameter	Data type	Description
b	*BLOB	Caller-owned BLOB; input for FromBlob, output for ToBlob.

Returns

No return value.

Before you call

A valid BLOB field in an open file is required. A BLOB is not a standalone local string variable. This code assumes DemoFile.Payload is such a field and a record is available.

Example

```
! Local Data
Text SystemStringClass
Other SystemStringClass
Result STRING(1024)

 CODE
Text.Str('cat')
Text.ToBlob(DemoFile.Payload)
Other.FromBlob(DemoFile.Payload)
Result = Other.Str()
```

Expected result

On a successful BLOB copy, Payload contains the bytes "cat" and Other contains "cat". Text remains "cat". Persist the record separately using the application's normal file handling.
The field name is illustrative; define it in your own file/dictionary. BLOB I/O has not been runtime verified.

12.4. ToFile

```
ToFile PROCEDURE(STRING fileName),BYTE,PROC
```

Description

Returns BYTE-declared status: zero success, BadFileErr or ERRORCODE on failure; do not interpret as boolean success.

Parameters

Parameter	Data type	Description
fileName	STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

BYTE — 0 on success; BadFileErr or a DOS driver error on failure.

Before you call

Choose a disposable, writable file path. ToFile writes it; FromFile requires a file containing the bytes to read. This example creates its own input.

Example

```
! Local Data
Text SystemStringClass
Other SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('cat')
Number = Text.ToFile('SystemString-doc-example.tmp')
IF Number = 0
 Other.Str('old')
 Number = Other.FromFile('SystemString-doc-example.tmp')
 IF Number = 0
 Result = Other.Str()
 END
END
```

Expected result

If both statuses are 0, the file bytes and Result are "cat". Text remains "cat"; Other becomes "cat" after reading. The example leaves the .tmp file in the current working directory.
File creation and driver behavior require runtime verification. Use a disposable directory; ToFile can replace existing content.

Notes

Returns zero on success. An empty source returns success without creating or truncating a file.

13. Capacity and append position

Select a method below. Detailed source evidence is available through each method’s Technical notes link.
GetAppendOnCursor
GetClipOnCursor
GetCursor
Length
Resize
SetAppendOnCursor
SetClipOnCursor
SetCursor
SetLen
SetLength

13.1. GetAppendOnCursor

```
GetAppendOnCursor PROCEDURE(),BYTE
```

Description

Returns the corresponding stored BYTE configuration flag.

Parameters

None.

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

No string needs to be loaded to read or change this setting. Configure it before the operation that uses it.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Number = Text.GetAppendOnCursor()
```

Expected result

New-object result: 0 (FALSE).

13.2. GetClipOnCursor

```
GetClipOnCursor PROCEDURE(),BYTE
```

Description

Returns the corresponding stored BYTE configuration flag.

Parameters

None.

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

No string needs to be loaded to read or change this setting. Configure it before the operation that uses it.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.SetClipOnCursor(TRUE)
Number = Text.GetClipOnCursor()
```

Expected result

Returned value: 1 (TRUE). This flag alone does not enable cursor appending.

13.3. GetCursor

```
GetCursor PROCEDURE(),LONG
```

Description

Returns appendCursor without normalization.

Parameters

None.

Returns

LONG — count or current position.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('cat')
Number = Text.GetCursor()
```

Expected result

Returned value: 4.
Stored string: "cat" (unchanged).

13.4. Length

```
Length PROCEDURE(),LONG
```

Description

Returns LEN(s), or 0 for NULL.

Parameters

None.

Returns

LONG — count or current position.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Number LONG

 CODE
Text.Str('cat ')
Number = Text.Length()
```

Expected result

Returned value: 5.
Stored string: "cat " (unchanged).

13.5. Resize

```
Resize PROCEDURE(LONG pValLen)
```

Description

Changes allocated main length, preserving the leading bytes and space-filling growth.

Parameters

Parameter	Data type	Description
pValLen	LONG	Requested byte length/count or width. See allocation and range restrictions.

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('cat')
Text.Resize(5)
Result = Text.Str()
```

Expected result

Stored string: "cat ". This call has no return value; Result reads the stored string afterward.

13.6. SetAppendOnCursor

```
SetAppendOnCursor PROCEDURE(BYTE value=true)
```

Description

Stores CHOOSE(value); enabling also enables clipping.

Parameters

Parameter	Data type	Description
value=true	BYTE	Base64 helper byte; reference parameters are outputs.

Returns

No return value.

Before you call

Allocate capacity before enabling cursor-based building. Enabling append mode also enables clip mode.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.NewString(8)
Text.SetAppendOnCursor(TRUE)
Text.Append('cat')
Result = Text.Str()
Number = Text.Length()
```

Expected result

Visible stored value: "cat". Allocated length: 8. Cursor is 4; hidden capacity remains.

13.7. SetClipOnCursor

```
SetClipOnCursor PROCEDURE(BYTE value=true)
```

Description

Stores CHOOSE(value).

Parameters

Parameter	Data type	Description
value=true	BYTE	Base64 helper byte; reference parameters are outputs.

Returns

No return value.

Before you call

Clip-on-cursor affects Str only when append-on-cursor is also enabled.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.NewString(8)
Text.SetAppendOnCursor(TRUE)
Text.Append('cat')
Text.SetClipOnCursor(FALSE)
Result = Text.Str()
```

Expected result

Result contains "cat " (five trailing spaces). Turning clipping back on would expose only "cat".

13.8. SetCursor

```
SetCursor PROCEDURE(LONG value)
```

Description

Accepts value when Length() >= value-1; zero maps to 1.

Parameters

Parameter	Data type	Description
value	LONG	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

No return value.

Before you call

Load or allocate storage first, then choose a cursor in 1..Length()+1. Enable cursor append to demonstrate it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('cat')
Text.SetAppendOnCursor(TRUE)
Text.SetCursor(2)
Text.Append('X')
Result = Text.Str()
```

Expected result

Visible stored value: "cX". Raw allocation contains "cXt"; cursor is 3 and clipping hides the final t.

13.9. SetLen

```
SetLen PROCEDURE(LONG pValLen)
```

Description

Replaces the entire main value with pValLen spaces; preserves previous appendCursor.

Parameters

Parameter	Data type	Description
pValLen	LONG	Requested byte length/count or width. See allocation and range restrictions.

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('cat')
Text.SetLen(5)
Result = Text.Str()
```

Expected result

Stored string: " ". This call has no return value; Result reads the stored string afterward.

13.10. SetLength

```
SetLength PROCEDURE(LONG pValLen)
```

Description

Replaces the entire main value with pValLen spaces; preserves previous appendCursor.

Parameters

Parameter	Data type	Description
pValLen	LONG	Requested byte length/count or width. See allocation and range restrictions.

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('cat')
Text.SetLength(5)
Result = Text.Str()
```

Expected result

Stored string: " ". This call has no return value; Result reads the stored string afterward.

14. Lifecycle and implementation details

Select a method below. Detailed source evidence is available through each method’s Technical notes link.
StringRefFactoryClass.Construct
StringRefFactoryClass.Destruct
StringRefFactoryClass.DisposeIt
StringRefFactoryClass.NewStringRef
AutoDispose
CleanSplitQueue
Construct
Destruct
DisposeIt
DisposeItCString
GetTrimmingLimits
Internalize
IsDelimiter
IsEOLMarker
Take24
Take32

14.1. StringRefFactoryClass.Construct

```
Construct PROCEDURE()
```

Description

Allocates StringRefQueue for tracking allocations.

Parameters

None.

Returns

No return value.

Before you call

Construct is automatic when NEW creates the factory. Do not call Construct again.

Example

```
! Local Data
Result STRING(1024)
Ref &STRING
Factory &StringRefFactoryClass

 CODE
Factory &= NEW StringRefFactoryClass
Ref &= Factory.NewStringRef(3)
Ref = 'cat'
Result = Ref
DISPOSE(Factory)
```

Expected result

Result contains "cat". Factory destruction releases its tracked allocation; Ref must not be used afterward.

14.2. StringRefFactoryClass.Destruct

```
Destruct PROCEDURE()
```

Description

Calls DisposeIt then disposes tracking queue.

Parameters

None.

Returns

No return value.

Before you call

Allocate a factory and obtain a factory-owned buffer before demonstrating automatic destruction.

Example

```
! Local Data
Result STRING(1024)
Ref &STRING
Factory &StringRefFactoryClass

 CODE
Factory &= NEW StringRefFactoryClass
Ref &= Factory.NewStringRef(3)
Ref = 'cat'
Result = Ref
DISPOSE(Factory)
```

Expected result

Before destruction the buffer contains "cat". DISPOSE(Factory) invokes Destruct and releases the buffer. No string is returned.

14.3. StringRefFactoryClass.DisposeIt

```
DisposeIt PROCEDURE()
```

Description

Disposes every tracked non-null STRING, then FREEs queue records.

Parameters

None.

Returns

No return value.

Before you call

Obtain a factory-owned buffer first. DisposeIt releases tracked buffers while retaining the factory.

Example

```
! Local Data
Result STRING(1024)
Ref &STRING
Factory &StringRefFactoryClass
```

```
 CODE
Factory &= NEW StringRefFactoryClass
Ref &= Factory.NewStringRef(3)
Ref = 'cat'
Result = Ref
Factory.DisposeIt()
DISPOSE(Factory)
```

Expected result

The copied Result remains "cat". Ref is invalid after DisposeIt; do not dereference or separately dispose it.

14.4. StringRefFactoryClass.NewStringRef

```
NewStringRef PROCEDURE(ULONG sizeValue),*STRING
```

Description

Allocates NEW STRING(sizeValue), ADDs reference to garbageStrs and returns it.

Parameters

Parameter	Data type	Description
sizeValue	ULONG	Requested byte length/count or width. See allocation and range restrictions.

Returns

Borrowed STRING reference, retained by the factory until disposal.

Before you call

Create a factory first; a main SystemString value is not involved.

Example

```
! Local Data
Result STRING(1024)
Ref &STRING
Factory &StringRefFactoryClass

 CODE
Factory &= NEW StringRefFactoryClass
Ref &= Factory.NewStringRef(3)
Ref = 'cat'
Result = Ref
DISPOSE(Factory)
```

Expected result

NewStringRef returns a factory-owned STRING reference. After assignment its contents are "cat". Do not dispose Ref separately.

Notes

Returns scratch storage, not the main string. The next scratch allocation invalidates the reference.

14.5. AutoDispose

```
AutoDispose PROCEDURE(*STRING newString),PROTECTED
```

Description

Protected: disposes previous scratch storage and adopts newString as garbageStr for later disposal.

Parameters

Parameter	Data type	Description
newString	*STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

PROTECTED.

Before you call

Protected method: the following statements belong inside a method of a derived SystemStringClass. Supply a NEW-allocated STRING, not a local fixed STRING.

Example

```
! Local Data
Result STRING(1024)
Ref &STRING
```

For protected examples, place the statements in a derived-class method and declare the listed working variables there. SELF is the derived object.

```
 CODE
Ref &= NEW STRING(3)
Ref = 'cat'
SELF.AutoDispose(Ref)
Result = Ref
```

Expected result

Result contains "cat". SELF owns the adopted scratch allocation; its main string is unchanged. Do not dispose Ref separately.

14.6. CleanSplitQueue

```
CleanSplitQueue PROCEDURE(*SystemStringQueue collection)
```

Description

Disposes every referenced SystemStringClass in the queue and FREEs queue records.

Parameters

Parameter	Data type	Description
collection	*SystemStringQueue	Queue whose referenced child objects will be disposed.

Returns

No return value.

Before you call

Load the source text, then Split it to populate the separate line queue. Line edits do not change the stored string until FromLines or Merge rebuilds it.

Example

```
! Local Data
Text SystemStringClass
Number LONG
QueueRef &SystemStringQueue

 CODE
Text.Str('red,green,blue')
Text.Split(',')
QueueRef &= Text.GetLines()
Text.CleanSplitQueue(QueueRef)
Number = Text.CountLines()
```

Expected result

Line count: 0; child objects were disposed. Queue allocation remains owned by Text. Main string remains "red,green,blue".

14.7. Construct

```
Construct PROCEDURE()
```

Description

Initializes the object.

Parameters

None.

Returns

No return value.

Before you call

Construct is automatic when a local object comes into scope or NEW allocates one.

Example

```
! Local Data
Result STRING(1024)
Owner &SystemStringClass

 CODE
Owner &= NEW SystemStringClass
Owner.Str('cat')
Result = Owner.Str()
DISPOSE(Owner)
```

Expected result

Result contains "cat". The constructor runs at NEW and the destructor runs at DISPOSE.

14.8. Destruct

```
Destruct PROCEDURE()
```

Description

Calls DisposeIt, releases the cached CSTRING, queued segments, line objects, line queue and scratch string.

Parameters

None.

Returns

No return value.

Before you call

Use DISPOSE on an object created with NEW. Do not call Destruct manually.

Example

```
! Local Data
Result STRING(1024)
Owner &SystemStringClass

 CODE
Owner &= NEW SystemStringClass
Owner.Str('cat')
Result = Owner.Str()
DISPOSE(Owner)
```

Expected result

Before disposal the stored string is "cat". DISPOSE releases the object and its owned storage. The copied Result remains "cat".

14.9. DisposeIt

```
DisposeIt PROCEDURE()
```

Description

Disposes the main STRING unless alreadyDisposed is set, resets token selection when s is non-null, then clears alreadyDisposed.

Parameters

None.

Returns

No return value.

Before you call

Load the text with Str (or SetString/FromString) first. The example starts with a new local object; its constructor runs automatically.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
Number LONG

 CODE
Text.Str('cat')
Text.DisposeIt()
Result = Text.Str()
Number = Text.Length()
```

Expected result

Result is empty; Number is 0. The Text object still exists.

14.10. DisposeItCString

```
DisposeItCString PROCEDURE()
```

Description

Disposes the cached CSTRING if non-null.

Parameters

None.

Returns

No return value.

Before you call

Load non-NULL text and call ToCString to create the cache before releasing it.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)
CRef &CSTRING

 CODE
Text.Str('cat')
CRef &= Text.ToCString()
Result = CRef
Text.DisposeItCString()
```

Expected result

Result contains "cat". The CRef cache reference is invalid after DisposeItCString. Stored string remains "cat".

14.11. GetTrimmingLimits

```
GetTrimmingLimits PROCEDURE(*LONG idxToTrimStart, *LONG idxToTrimEnd, BYTE trimType, STRING char1,
<STRING char2>, <STRING char3>, <STRING char4>),BYTE,PROC,PROTECTED
```

Description

trimType: 0 both, 1 left, 2 right.

Parameters

Parameter	Data type	Description
idxToTrimStart	*LONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.
idxToTrimEnd	*LONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.
trimType	BYTE	0 both ends; 1 left; 2 right.
char1	STRING	Trim candidate; only first byte is used.
char2	STRING	Trim candidate; only first byte is used. Optional (omittable).
char3	STRING	Trim candidate; only first byte is used. Optional (omittable).
char4	STRING	Trim candidate; only first byte is used. Optional (omittable).

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

Protected: use these statements inside a derived-class method. Load a nonempty value containing at least one retained byte; bounds are passed by reference.

Example

```
! Local Data
Result STRING(1024)
Number LONG
FirstBound LONG
LastBound LONG
```

For protected examples, place the statements in a derived-class method and declare the listed working variables there. SELF is the derived object.

```
 CODE
SELF.Str('--cat--')
Number = SELF.GetTrimmingLimits(FirstBound, LastBound, 0, '-')
Result = SELF.Str()
```

Expected result

Returned flag: 1. FirstBound is 2; LastBound is 6. Retained bytes are 3..5 ("cat"). Stored string remains "--cat--".

14.12. Internalize

```
Internalize PROCEDURE(*STRING svalue),PROTECTED
```

Description

Protected: disposes old main storage and adopts svalue by reference, resets wasExternalized.

Parameters

Parameter	Data type	Description
svalue	*STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

PROTECTED.

Before you call

Protected method: use these statements inside a derived-class method. Transfer a NEW-allocated STRING to the object.

Example

```
! Local Data
Result STRING(1024)
Ref &STRING
```

For protected examples, place the statements in a derived-class method and declare the listed working variables there. SELF is the derived object.

```
 CODE
Ref &= NEW STRING(3)
Ref = 'cat'
SELF.Internalize(Ref)
Result = SELF.Str()
```

Expected result

Stored string and Result are "cat". SELF owns Ref afterward; the previous main allocation was released. Cursor is not repaired by Internalize.

14.13. IsDelimiter

```
IsDelimiter PROCEDURE(LONG checkIndex),BYTE,PROTECTED
```

Description

Protected helper operating on raw storage.

Parameters

Parameter	Data type	Description
checkIndex	LONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

Protected: these statements belong inside a derived-class method. Load text and configure delimiters before testing a valid 1-based byte index.

Example

```
! Local Data
Number LONG
```

For protected examples, place the statements in a derived-class method and declare the listed working variables there. SELF is the derived object.

```
 CODE
SELF.Str('A,B')
SELF.SetTokensDelimiters(',')
Number = SELF.IsDelimiter(2)
```

Expected result

Returned value: 1 (TRUE), because byte 2 is comma. Stored text remains "A,B".

14.14. IsEOLMarker

```
IsEOLMarker PROCEDURE(LONG checkIndex),BYTE,PROTECTED
```

Description

Protected helper operating on raw storage.

Parameters

Parameter	Data type	Description
checkIndex	LONG	Position or segment boundary; normally 1-based. See method-specific boundary rules.

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

Protected: use inside a derived-class method. Set the EOL marker and choose a valid index with enough following storage for the implementation check.

Example

```
! Local Data
Number LONG
```

For protected examples, place the statements in a derived-class method and declare the listed working variables there. SELF is the derived object.

```
 CODE
SELF.Str('A|B')
SELF.SetEOLMarker('|')
Number = SELF.IsEOLMarker(2)
```

Expected result

Returned value: 1 (TRUE). Stored text remains "A|B". The boundary check can reject a marker at the end; see technical notes.

14.15. Take24

```
Take24 PROCEDURE(byte h, byte m, byte l, *STRING Into),PRIVATE
```

Description

Private Base64 helper.

Parameters

Parameter	Data type	Description
h	byte	Base64 helper byte; reference parameters are outputs.
m	byte	Base64 helper byte; reference parameters are outputs.
l	byte	Base64 helper byte; reference parameters are outputs.
Into	*STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

PRIVATE.

Before you call

Private helper: application code cannot call Take24. Exercise it through EncodeBase64 or ToBase64 instead.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('cat')
Result = Text.ToBase64()
```

Expected result

Public-path result: "Y2F0". Stored string remains "cat". Internally Take24 converts a three-byte group to four Base64 characters.

14.16. Take32

```
Take32 PROCEDURE(*byte h, *byte m, *byte l, *STRING SFrom),BYTE,PRIVATE
```

Description

Private Base64 helper.

Parameters

Parameter	Data type	Description
h	*byte	Base64 helper byte; reference parameters are outputs.
m	*byte	Base64 helper byte; reference parameters are outputs.
l	*byte	Base64 helper byte; reference parameters are outputs.
SFrom	*STRING	Input text, object, delimiter or configuration value as used in behavior notes.

Returns

BYTE — stored flag or TRUE/FALSE status; see Technical notes for exact conditions.

Before you call

Private helper: application code cannot call Take32. Exercise it through DecodeBase64 instead.

Example

```
! Local Data
Text SystemStringClass
Result STRING(1024)

 CODE
Text.Str('Y2F0')
Text.DecodeBase64()
Result = Text.Str()
```

Expected result

Public-path stored result: "cat". Internally Take32 decodes a four-character Base64 group. The private BYTE return is not exposed here.

15. Alphabetical API index

StringRefFactoryClass.Construct
StringRefFactoryClass.Destruct
StringRefFactoryClass.DisposeIt
StringRefFactoryClass.NewStringRef
AddInsertInContentSegment
AddLine
AddReplaceInContentSegment
Append
AppendLine

AutoDispose
Clean
CleanLines
CleanSegments
CleanSplitQueue
Compress
Construct
Contains
CountLines
CountToken
CountTokens
CountWord
CountWords
DecodeBase64
Decompress
DeleteLine
Destruct
DisposeIt
DisposeItCString
EncodeBase64
EndsWith
Equals
Escape
Externalize
ExtractKeyValue
FormatString
FoundToken
FromBase64
FromBlob
FromFile
FromLines
FromLinesWithEOL
FromString
GetAppendOnCursor
GetCaseSensitive
GetClipOnCursor
GetCurrentToken
GetCursor
GetEOLMarker
GetLeftOf
GetLine
GetLineTrimValue
GetLineValue
GetLines
GetLinesCount
GetPairSeparator
GetRightOf
GetString
GetStringRef
GetSurrounded
GetToken
GetTokensDelimiters
GetTrimmingLimits
GetUrlEncodeDataChar

GetUseLowerAsciiAsDelimiters
GetValueSeparator
GetWord
HtmlEscape
IndexOf
IndexOfReg
Insert
InsertInContentSegment
InsertLine
Internalize
IsDelimiter
IsEOLMarker
IsExternalized
IsPastEOL
IsTokenEmpty
IsTokenInited
IsTokenNull
LastIndexOf
Length
Merge
NewString
NewStringRef
NextToken
Overwrite
PadCenter
PadLeft
PadRight
PopToken
Prepend
PushToken
Remove
RemoveSubString
RemoveSubStringInContent
Replace
ReplaceInContent
ReplaceInContentSegment
ReplaceKeyValue
ReplaceSubString
ResetToken
Resize
SetAppendOnCursor
SetCaseSensitive
SetClipOnCursor
SetCursor
SetEOLMarker
SetLen
SetLength
SetLineValue
SetPairSeparator
SetPairValueSeparator
SetString
SetSubstring
SetTokensDelimiters
SetUrlEncodeDataChar

SetUseLowerAsciiAsDelimiters
SetValueSeparator
Split
SplitToLines
StartsWith
Str
Substring
SubstringLeft
Surround
Take24
Take32
ToBase64
ToBlob
ToCString
ToCapitalize
ToCapitalizeInContext
ToFile
ToLower
ToLowerInContext
ToString
ToUpper
ToUpperInContext
Trim
TrimEnd
TrimInContext
TrimStart
TryGetLineValue
UnEscape
UrlDecode
UrlEncode

16. Categories overview

Choose a category to find functions by task.

Build and edit strings
Capacity and append position
Change case, trim and pad
Encode, decode and compress
Create, assign and clear
Extract portions
Files and BLOBs
Format and process key/value pairs
Lifecycle and implementation details
Split, edit and join lines
Search and compare
Tokens and words
Read values and manage references