CREXX
Library Reference

The CREXX team

September 14, 2026

THE REXX LANGUAGE ASSOCIATION
CREXX Programming Series
ISBN 978-94-039116-4-9

Publication Data

The responsible publisher of this edition is identified as *IBizz IT Services and Consultancy*, Amsteldijk 14, 1074 HR Amsterdam, a registered company governed by the laws of the Kingdom of The Netherlands.

This edition is registered under ISBN 978-94-039116-4-9

ISBN 978-94-039116-4-9

9 789403 911649 >

Content is up to date with version *crexx-1.0.0-beta.3+local.g4c4f4c48643e*

Contents

The CREXX Publication Series

This book is part of a library, the CREXX *Programming Series*, documenting the CREXX programming language and its use and applications. This section lists the other publications in this series, and their roles. These books can be ordered in convenient hardcopy and electronic formats from the Rexx Language Association.

Programming Guide	The Programming Guide explains the working of the tools and has examples for building programs on the platforms it supports.
Language Reference	Referred to as the CRL, this is meant as the formal definition for the language, documenting its syntax and semantics, and prescribing minimal functionality for language implementers.
Library Reference	A complete reference of all included functionality in the CREXX distribution.
VM Specification	The CREXX VM Specification documents the CREXX Assembly Language and its execution by the CREXX Virtual Machine. It also contains low level virtual machine and ABI specifications.

Typographical conventions

In general, the following conventions have been observed in the CREXX publications:

- Body text is in this font
- Examples of language statements are in a `keyword` or **bold** type
- Items that are introduced, or emphasized, are in an *italic* type
- Included program fragments are listed in this fashion:

```
/* salute the reader */
say 'hello reader'
```

Console output generated from programs contained in the text is shown in this form:

```
hello reader
```

1 About the Library Reference

This publication contains the documentation of the CREXX standard library. All documented functionality is delivered in every distribution of the CREXX system, while platform dependent differences or supersets are relegated to the appendices. The REXX built-in functions, traditionally seen as part of the language, are also documented in the *Language Reference*, but in this *Library Reference* there is an emphasis on platform dependent functions and CREXX specific additions.

With Classic REXX having its origins as a language intended for command processing and procedural programming, two object oriented successors, Object REXX and NetREXX have introduced (divergent) class concepts, types and syntax. CREXX introduced native hardware types[1] and an object oriented notation which stays closer to Classic REXX (with standard labels and `class` and `interface` keywords in the location of the `procedure` keyword. In cRexx, the `stem` type is implemented as a class, part of this library, and useable in procedural programs as any other type. In addition to this, a large number of other data structures are documented here, and are considered part of the runtime library instead of components of the core language. This has enabled the CREXX team to deliver a set of data structures recognisable to users of both predecessors.

This level of the Library does not use inheritance and polymorphic calls (overloaded signatures). If and when this is the case, this message will be removed and method names will be losing type info. Removal of typed method names will have

[1] NetRexx types are still Java VM types but correspond to, and are implemented as generic hardware concepts.

a deprecation period. The current APIs cannot be regarded as stable in this regard. To avoid a large change, interfaces that accept '.string' type will leave out the type name and will be the stable name.

Also to be found here are native implementations of Graphic User interfaces and database drivers, both *universal* like ODBC or specific drivers for engines like *sqlite* - next to CREXX implementations of keystores or interface libraries.

An explicit design goal was to keep the standard library structure relatively flat, i.e. without deep hierarchies which need to be included and searched, and which make navigation of the library source and comparison of like classes complicated. The categories of the library components are intended to limit the search scope fairly quickly.

The top level of the library is:

- System
- Data
- Time
- GUI
- Math

The current release status of this Standard Library is that it is preliminary reference material rather than a part of the Release 1 beta baseline language contract. The user has to verify components, modules and drivers before depending on these API's. All documented API's have implementations - planned functionality in here is clearly marked. Although great care will be taken to keep the API contracts stable, there is, however, no guarantee for that at this time.

PART I

Objectives

2

Goals

An implicit goal of this class library is to take what other Rexx implementations have to offer as a starting point. Classic Rexx extended and supplanted simple command list processors and added procedural structure, simplification, dropped unneeded punctuation and made lowercase available. An important addition was a string-based structured data type, the `stem` type, which implemented a content-addressable multidimensional array. In release 2 Classic Rexx already added unlimited decimal precision. Subsequent language variants kept these Rexx achievements and added more types: Object Rexx ('Oryx') added an elegant Smalltalk-inspired class hierarchy with mixin multiple inheritance, while NetRexx reused the JVM class library by offering a seamless interface, and added the JVM native machine types to Rexx, while offering single inheritance and interface inheritance. All collection classes are the JVM's. It is the intention of the crexx team to have a large selection of these API shapes available in a crexx environment; this implies that a certain amount of duplication is unavoidable. This is less problematic than ever and we prefer choice over sparseness. This enables a running start for Classic Rexx, NetRexx and Object Rexx users. It also opens perspectives for mutual reinforcement; like when Object Rexx users who never had a collection with ordered inserts, might use the TreeMap class now. We try to deliver compatible OS and math implementations, and for example known libraries like `RexxUtil` can be find in addition to the crexx OS level support. But compatibility is no goal in itself.

This library also has a number of explicit goals:

- Familiarity: it is reasonable to expect that the programmer has experience with several class libraries, and as such there can be expectations of what is included in a standard class library; the core Rexx API's are available in both procedural and object-oriented notation;

- Ease of reference: a class should be findable where the programmer first looks for it; the information on the class API should be to the point and complete;
- Fit for complex purposes: Design of class structures for the application domain is best left to the application designer; this library aims to offer working and quickly applicable solutions for the building blocks of application systems, like parsing, data access like json, database or network access. A simple parser is easily written in the core crexx language; the special format parsers like the beforementioned offer correctness and performance for some of these deceptively complex protocols, a generic ODBC driver for a large range of relational databases is an asset for every data oriented application;
- Performance: the aim has been to provide implementations of classes which have the highest level of performance possible; this to complement the work done on compiler optimisation, inlining of code, profiling and benchmarking. The emphasis is on native crexx implementations which are optimally suited to profit of this work. This is the reason that there is less emphasis on reusing available libraries for other programming environments via bridges or marshalling. Preferably, native implementations are offered.

2.1 Note about implementation

The contents of the library are put together using different technologies, although great care has been taken, using the `namespace` and `import` mechanisms, to hide that fact from the user. The crexx class library is implemented primarily in the crexx language itself, and when that was impossible because of instruction set architecture or operating system dependencies, using plugins in the C language. So a library is just a library until it is not: it can turn out to be a portable `.rxbin` library, or a native statically linked or dynamically linkable shared object library or dll. Only in large cross-platform applications, packaging of statically linked object libraries versus shared object libraries becomes slightly more complex. In most cases, an optional performance increase brings a modest price in complexity.

All components can be re-packaged with the `rxlink` utility, while all the modules and libraries can be completely rebuilt from a source distribution or a clone of the CREXX git repository. The *Programming Guide* has a chapter on build systems that are of use when building large heterogeneous applications.

3

Class library structure

As mentioned, the intention is to leave the naming structure relatively flat. The reason for this is to limit the need for convoluted import statements which are hard to remember and take up a lot of space in program source files. It should be easy to remember which part of the library to include and the decision to determine from which part of the library a specific class is should be straightforward. The currently active parts are:

TABLE 1: Library Namespaces.

Namespace	Domain	Example Classes
System	OS related functionality; things that are dependent or specific for the OS	beep, wait, listdir,
Data	Collection classes and data interchange (I/O and formats)	Sets, Maps, Lists, Stems, Arrays, json, xml, yaml, ODBC
Time	Time, Date, Calender oriented functionality	Time, Date, Calendar, ISO Date, Locale Calendars
Math	Mathematics functions (Trigonometry, Statistics, Conversions)	sin, cos, mean, etc.
GUI	Graphical User Interfaces, Text based or bitmapped	curses, GTK

PART II

Overview

4 Collection Classes

The collection classes provide an object-oriented idiom for handling (adding, deleting, sorting, searching) of data. All are part of the data namespace. They implement interfaces like `.Iterable`, `.Iterator`, `.List`, `.Map` and `.Tree`. In this chapter these interfaces will be discussed and explained; the details are in the reference section of this book.

4.1 The interfaces

These interfaces distinguish between three ways of managing data.

- **List** — an ordered sequence of elements.
- **Set** — a collection of unique elements.
- **Map** — a collection of key/value associations.

A useful analogy is:

- List → a shopping list
- Set → a bag of unique stamps
- Map → a dictionary

`Map` stores *pairs* rather than individual objects.

4.2 List

A `List` is an ordered collection of elements. Unlike a `Set`, which is concerned solely with membership, a list preserves the position of every element and allows the same value to appear more than once. In many respects, a list corresponds to the intuitive notion of a sequence: the order in which elements are inserted is significant, and each element occupies a well-defined position within that sequence.

The concept of a list predates modern programming languages by many decades. In mathematics, ordered collections appear as sequences, tuples, and vectors, where the arrangement of the elements is as important as the elements themselves. The sequence *(2, 3, 5)* is fundamentally different from *(5, 3, 2)*, even though both contain the same values. Computer science adopted this notion early on, and lists became one of its most important abstract data types. They form the basis of countless algorithms and data structures, from symbol tables and compiler token streams to document models and graphical user interfaces.

The defining characteristic of a list is that every element has a position, usually referred to as its **index**. In cRexx, indexing begins at one, so the first element has index `1`, the second index `2`, and so forth. Because every element occupies a unique position, a list provides efficient access to the *n*th element, making it possible to retrieve or replace an element directly by its index. Unlike a set, duplicate elements are perfectly acceptable; if the same value occurs several times, each occurrence occupies its own position in the sequence.

The operations provided by the `List` interface revolve around this notion of ordered storage. Elements may be appended to the end of the list or inserted at an arbitrary position, causing subsequent elements to shift to make room. Existing elements can be replaced without altering the size of the list, or removed entirely, in which case later elements move forward to close the gap. Lists also support searching for a value, determining the size of the collection, and traversing the elements in their stored order. Because ordering is intrinsic to the abstraction, iteration always proceeds from the first element to the last unless explicitly reversed.

An important consequence of positional storage is that a list preserves duplicates exactly as they were inserted. Consider a compiler that records every identifier encountered in a source file. If the identifier `count` appears twenty-three times, a list faithfully records all twenty-three occurrences in their original order. A set, by contrast, would retain only a single instance because its purpose is to represent the unique identifiers appearing in the program rather than every occurrence.

Although the `List` interface defines a common abstraction, different implementations are optimized for different patterns of use. The most widely used implementation is `ArrayList`, which stores its elements in a dynamically resized array. Because the elements occupy contiguous memory locations, random access by index is extremely efficient and requires constant time. Appending new elements to the end of the list is likewise very efficient, with only occasional resizing of the underlying array. Inserting or removing elements in the middle of the list, however, requires subsequent elements to be shifted, making these operations proportional to the size of the list.

`LinkedList` takes a fundamentally different approach by storing each element in a separate node connected by references to its neighbours. This representation makes insertions and removals at either end of the list, or at a location already reached by an iterator, very efficient because only a small number of references must be updated. Random access, however, is considerably slower, since locating the *n*th element requires traversing the list from one end or the other. Consequently, a linked list is most appropriate when the application performs frequent insertions and deletions but comparatively little indexed access.

`CREXX` also provides a `Stack` class. It implements the familiar last-in, first-out behaviour.

Several useful operations are defined specifically for lists because of their ordered nature. A list may be traversed in either direction using a `ListIterator`, sorted according to the natural ordering of its elements or a supplied comparator, reversed, or partitioned into a view representing a contiguous range of elements. Searching operations can locate the first or last occurrence of a value, reflecting the fact that duplicate elements may exist at multiple positions. These operations have no meaningful counterparts for sets, where the absence of ordering makes concepts such as "first" and "last" undefined.

Lists are among the most versatile data structures in software development and are frequently used whenever information has a natural sequence. Examples include the lines of a source program, the tokens produced by a lexical analyser, the statements of an abstract syntax tree, the history of commands entered into a shell, the items displayed in a graphical menu, or the chapters of a book. In each of these cases, the order of the elements carries essential information that would be lost if they were stored in an unordered collection.

In practice, a `List` should be chosen whenever the relative order of elements is significant or when access by position is an important requirement. If duplicates must be preserved, if the *n*th element must be retrieved efficiently, or if the collection naturally represents a sequence rather than merely a set of distinct values, a `List` is the appropriate abstraction. The central question answered by a list is therefore not simply *whether* an element exists, but also *where* it occurs within the sequence. That emphasis on order and position distinguishes the `List` interface from every other collection type in CREXX class library.

4.2.1 Summary

A `List` is an ordered collection.

Properties:

- preserves insertion order
- allows duplicate elements
- elements have an index
- random access may or may not be efficient depending on implementation

Example:

```
["apple", "banana", "apple", "pear"]
```

Duplicates are perfectly legal.

You can retrieve by position:

```
list.get(0)   -> "apple"
list.get(2)   -> "apple"
```

Typical operations

```
add(E e)
add(int index, E e)

get(int index)

set(int index, E e)

remove(int index)
```

```
contains(E e)

size()
```

Typical implementations

```
ArrayList
LinkedList
Stack (legacy)
```

4.2.2 Complexity

For `ArrayList`

```
get(index)        O(1)

append            O(1) amortized

insert middle     O(n)

remove middle     O(n)
```

For `LinkedList`

```
get(index)        O(n)

insert front      O(1)

remove front      O(1)
```

CREXX also offers a `Stack` class which actually is limited form of a `list`, in which elements can only be added and removed from one end. It has `push` and `pop` methods but no `get`. A stack follows the LIFO (Last In, First Out) principle. Most explanations of the `Stack` type conjure up a picture of a stack of plates in a cafeteria.

4.3 Set

A `Set` is a collection of **distinct elements**. Unlike a `List`, a set has no notion of position or index; what matters is whether an element is a member of the set, not where it appears. This seemingly simple idea has deep mathematical roots and has influenced many areas of computer science.

The concept of a set forms one of the foundations of modern mathematics. In the late nineteenth century, **Georg Cantor** developed set theory as a rigorous way of reasoning about collections of objects. Since then, sets have become fundamental to nearly every branch of mathematics. Computer science inherited these ideas, and many of its central concepts—including relations, functions, graphs, formal languages, and databases—can ultimately be described in terms of sets.

A mathematical set is simply a collection of distinct objects, called *elements* or *members*. The defining characteristic is that an element either belongs to the set or it does not; multiple occurrences of the same element have no meaning. Thus the sets `{2, 3, 5, 7}` and `{7, 2, 5, 3}` are considered identical because they contain exactly the same members, and `{2, 3, 3, 5, 7}` represents the very same set because duplicate elements are ignored.

The `Set` interface follows this mathematical model closely. When an element is added that is already present, the operation has no effect and the set remains unchanged. Consequently, the interface does not provide positional operations such as `get(int index)`, because the concept of an index is incompatible with the notion of a set. Instead, the fundamental operations revolve around membership: adding elements, removing them, testing whether a particular element is present, determining the number of elements, and iterating over the members.

One of the greatest strengths of the `Set` abstraction is that it naturally supports the operations of elementary set theory. The most familiar of these is the **union**, which combines two sets into a new set containing every element that appears in either of them. Equally important is the **intersection**, which retains only those elements that the two sets have in common. The **difference** of two sets consists of the elements that belong to one set but not the other, while the **symmetric difference** contains precisely those elements that occur in one set or the other, but not both.

Another important concept is that of a **subset**. A set A is said to be a subset of another set B if every element of A is also an element of B. If, in addition, the two sets are not identical, A is called a **proper subset** of B. Subset relationships occur frequently in computing, for example when verifying that a user possesses all required permissions, that one collection of compiler options is contained within another, or that the symbols used in an expression belong to a language's alphabet.

The CREXX class library provides three principal implementations of the `Set` interface, each optimized for different requirements. `HashSet` stores its elements in a hash table and offers constant-time performance for insertion, removal, and membership testing on average. Because hashing does not preserve any ordering, the iteration order of a `HashSet` is intentionally unspecified. It is therefore the preferred implementation whenever ordering is irrelevant and maximum performance is desired.

`LinkedHashSet` extends this design by maintaining a linked list of its elements in insertion order. This preserves the efficiency of hash-based lookup while ensuring that iteration visits the elements in the order in which they were originally added. The additional bookkeeping incurs only a modest memory overhead, making `LinkedHashSet` an attractive choice when predictable iteration order is required.

`TreeSet` takes a different approach by storing its elements in a self-balancing binary search tree, specifically an AVL tree[2] Rather than preserving insertion order, it maintains its elements in sorted order according to their natural ordering or a user-supplied comparator. Because the tree remains balanced, insertion, removal, and lookup all require logarithmic time. The ordered representation also enables a rich collection of navigation operations, including finding the smallest or largest element, locating the nearest element greater or less than a given value, and obtaining subsets that fall within a specified range. These capabilities make `TreeSet` particularly well suited for applications where maintaining sorted data is more important than achieving the absolute fastest lookup time.

In practice, a `Set` should be chosen whenever the uniqueness of elements is the primary concern. Typical applications include maintaining the collection of identifiers encountered during compilation, recording the set of visited nodes during graph traversal, representing the keywords of a programming language, storing the enabled options of a compiler, or eliminating duplicate entries from an input stream. In each of these cases, the essential question is not *where* an element occurs, but simply *whether* it is present. That distinction lies at the heart of the `Set` abstraction and explains why it occupies such a central place in both mathematics and CREXX class library.

4.3.1 Summary

A `Set` stores unique elements.

There is no concept of position.

[2] Knuth 1998

Example

```
{"apple", "banana", "pear"}
```

Adding another `"apple"` changes nothing.

```
set.add("apple")
```

returns

```
false
x */
```

because it already exists.

Typical operations

```
add(E e)

remove(E e)

contains(E e)

size()
```

Notice there is no

```
get(index)
```

because elements have no index.

4.4 HashSet

Internally based on a hash table.

Properties

- no duplicates
- very fast lookup
- iteration order unspecified

Typical complexity

```
add()          O(1)

contains()     O(1)

remove()       O(1)
```

4.5 LinkedHashSet

Same uniqueness guarantee.

Additionally

- preserves insertion order

```
add red
add green
add blue
```

Iteration always produces

```
red
green
blue
```

4.6 TreeSet

Internally a balanced binary search tree (specifically an AVL tree).

Properties

- unique elements
- automatically sorted

Example

Insert

```
pear
apple
orange
```

Iteration becomes

```
apple
orange
pear
```

Complexity

```
add()         O(log n)

contains()    O(log n)

remove()      O(log n)
```

4.7 Map

A `Map` is a collection that associates **keys** with **values**. Unlike a `List`, which stores elements in a sequence, or a `Set`, which stores a collection of unique elements, a map stores *relationships*. Each key identifies exactly one value, allowing the value to be retrieved efficiently whenever its corresponding key is known. In many programming languages this abstraction is also known as an *associative array*, a *dictionary*, or a *symbol table*.

The concept of a map is closely related to the mathematical notion of a **function**. In mathematics, a function assigns every element of one set, called the *domain*, to a single element of another set, called the *codomain*. While CREXX maps are generally more flexible than mathematical functions—they are mutable, need not define a value for every possible key, and may associate different keys with the same value—the underlying idea is remarkably similar. A map defines a correspondence between one collection of objects and another.

This distinction is important because a map is fundamentally different from the other collection interfaces. A list answers the question *"What is the element at position n ?"* A set answers *"Is this element present?"* A map answers *"Which value belongs to this key?"* Rather than storing isolated objects, a map stores associations, and those associations become the primary object of interest.

The defining characteristic of a map is that each key is unique. Attempting to insert a new key-value pair whose key already exists does not create a duplicate entry; instead, the new value replaces the old one. Values, however, need not be unique. Several different keys may legitimately refer to the same value. For example, a compiler may associate multiple identifiers with the same data type, or several employees may belong to the same department. The uniqueness constraint therefore applies only to the keys.

The operations provided by the `Map` interface reflect this model of association. New mappings can be inserted, existing mappings updated, and mappings removed entirely. Given a key, the associated value can be retrieved efficiently, or the map can be queried to determine whether a particular key or value is present. The interface also provides views of the map's keys, values, and entries, allowing the contents to be traversed in a variety of ways. These views illustrate the close relationship between maps and collections: while a map is not itself a `Collection`, it can expose its keys as a `Set`, its values as a `Collection`, and its key-value associations as a `Set` of `Map.Entry` objects.

From a mathematical perspective, maps support many operations analogous to those defined for functions and relations. Two maps may be compared for equality by determining whether they contain precisely the same key-value associations. A map may be copied, merged with another map, filtered according to its keys or values, or transformed by applying a function to each value.

The CREXX class library provides several implementations of the `Map` interface, each optimized for different requirements. The most frequently used implementation is `HashMap`, which stores its mappings in a hash table. Under normal circumstances, insertion, lookup, and removal all require constant time on average, making `HashMap` the preferred choice whenever maximum performance is desired and the order of the keys is unimportant. Because the placement of entries is determined by their hash codes, iteration order is intentionally unspecified and may vary as the map grows.

`LinkedHashMap` extends `HashMap` by maintaining a linked list of its entries. This additional structure preserves a predictable iteration order, usually the order in which mappings were inserted, while retaining nearly the same performance characteristics as a hash table. The class also supports access-order iteration, allowing recently accessed entries to migrate toward the end of the sequence. This feature makes `LinkedHashMap` particularly useful for implementing least-recently-used (LRU) caches and similar data structures.

`TreeMap` stores its entries in a self-balancing binary search tree, specifically an AVL tree. Rather than preserving insertion order, it maintains the keys in sorted order according to their natural ordering or a user-supplied comparator. As a consequence, insertion, removal, and lookup require logarithmic rather than constant time. In return, the map gains a rich collection of navigational operations, including finding the smallest or largest key, locating the nearest key greater or less than a given value, and obtaining views representing contiguous ranges of keys. These capabilities make `TreeMap` particularly attractive when ordered traversal or range queries are more important than raw lookup speed.

Maps are among the most frequently used abstractions in software engineering because many problems naturally involve associations rather than sequences. A compiler maintains a symbol table that associates identifiers with their declarations, an interpreter maps variable names to their current values, a database index relates primary keys to stored records, and a web server associates URL paths with request handlers. Configuration systems map property names to configuration values, while caches associate computed results with the requests that produced them. In every case, the central concern is not the order in which objects were inserted, nor merely whether they exist, but the relationship between one object and another.

In practice, a `Map` should be chosen whenever information is naturally expressed as a collection of key-value associations. If each object has a unique identifier, if rapid lookup by that identifier is essential, or if the problem itself is fundamentally one of representing relationships, a map is almost always the appropriate abstraction. Whereas a `List` organizes information by position and a `Set` organizes it by membership, a `Map` organizes it by correspondence. That ability to model associations efficiently and naturally explains why the `Map` interface occupies such a central place in the CREXX class library.

A `Map` associates keys with values, as a dictionary does.

```
"NL" -> "Netherlands"

"UK" -> "United Kingdom"

"D" -> "Germany"
```

Each key is unique, but values need not be unique.

Example

```
Peter -> Muenchen

Adrian -> Birmingham

René -> Amsterdam

Rony -> Vienna

Michael -> Vienna
```

The city appears more than once.

Typical operations

```
put(K key, V value)

get(K key)
```

```
remove(K key)
```

```
containsKey(K key)
```

```
containsValue(V value)
```

```
keySet()
```

```
values()
```

```
entrySet()
```

Example

```
map.put("apple", 5);
```

```
map.put("banana", 8);
```

```
map.get("apple")
```

returns

```
5
 rexx */
```

If

```
map.put("apple", 12)
```

the previous value is replaced.

Result

```
apple -> 12
banana -> 8
```

4.8 HashMap

Internally a hash table.

Complexity

`put()`	`O(1)`
`get()`	`O(1)`
`remove()`	`O(1)`

Order is unspecified.

4.8.1 ObjectObjectHashMap

`.ObjectObjectHashMap(strategy, expected)` is the concrete class-library map for object keys and object values when equality and hashing are domain policy. Its `.ObjectKeyStrategy` supplies both `hash(key)` and `equivalent(left, right)`; equivalent keys must always have the same hash.

The implementation caches hashes, resolves collisions by calling the strategy, normalizes negative hashes, and resizes its separate-chaining table. Key and value snapshots, including `iterator()`, preserve insertion order. Replacing an equivalent key retains the originally stored key and its position; removing and reinserting a key appends it at the end.

There is no default identity strategy. Raw object addresses, reference-cell identity, and cross-worker live references are not map-key contracts. Use `<refsame>` separately when code explicitly needs to compare two reference values, or define immutable logical handle fields for a genuinely global resource.

4.9 LinkedHashMap

Like `HashMap` but preserves insertion order.

```
A
B
C
x */
```

will iterate

```
A
B
C
x */
```

4.10 TreeMap

Keys are automatically sorted.

Insert

```
orange
```

```
pear
apple
```

Iteration

```
apple
orange
pear
```

Complexity

`put()`	`O(log n)`
`get()`	`O(log n)`
`remove()`	`O(log n)`

4.11 Comparing List, Set and Map

Feature	List	Set	Map
Stores	elements	unique elements	key/value pairs
Duplicate elements	Yes	No	Keys: No, Values: Yes
Ordered	Yes	Depends	Depends
Indexed	Yes	No	No
Lookup by key	No	No	Yes
Sorted variant	No	TreeSet	TreeMap

4.12 Choosing the Right Interface

Use a `List` when:

- order matters
- duplicates are allowed
- you need positional access

Example

```
shopping list

chapter list

playlist
```

Use a `Set` when:

- uniqueness matters
- fast membership testing is important

Example

```
keywords
```

```
visited URLs
```

```
registered usernames
```

Use a `Map` when:

- every object has an identifier
- you need fast lookup by key

Example

```
employee ID -> employee
```

```
word -> definition
```

```
country code -> country name
```

4.13 Relationships between interfaces

Suppose you have

```
a = HashMap()
```

Internally it conceptually stores entries such as

```
("apple",5)
```

```
("banana",8)
```

```
("pear",2)
```

From this map you can obtain:

```
keySet()
/
```

which is a `Set of .string`

```
apple
banana
pear
```

or

```
values()
/
```

which is an array of integers.

```
5
8
2
x */
```

or

```
entrySet()
```

which is a `Set` of keys

```
(apple,5)

(banana,8)

(pear,2)
```

Thus `Map` provides collection views of its keys, values, and entries.

4.14 Summary

```
List
    Ordered sequence
    Duplicates allowed
    Indexed access

Set
    Unique elements
    No indexing
    Fast membership testing

Map
    Key/value associations
    Unique keys
    Fast lookup by key
```

A good rule of thumb is:

- If you ask "What is the *nth* item?" use a `List`.
- If you ask "Have I seen this item before?" use a `Set`.
- If you ask "Given this key, what value belongs to it?" use a `Map`.

4.15 The Iterable interface

The `Iterable` interface is the smallest and most fundamental abstraction in the CREXX class library. It represents nothing more than the ability to traverse a collection of objects one element at a time. Any class that implements `Iterable` promises that its contents can be visited sequentially, regardless of how those contents are actually stored.

Unlike `List`, `Set`, or `Map`, `Iterable` says nothing about ordering, uniqueness, indexing, or searching. It merely states that there exists a way to obtain an iterator capable of visiting every element in the object. This modest contract makes `Iterable` one of the cornerstones, as it provides a common mechanism for traversing virtually every CREXX collection.

Conceptually, `Iterable` separates **traversal** from **representation**. A collection may internally be implemented as an array, a linked list, a balanced tree, a hash table, or even a structure that generates elements on demand. As long as it can produce an iterator, clients need not know or care about its internal organization.

The interface itself is remarkably small. Its only required method is `iterator`. As an example:

```
options levelb comments_dash
namespace data expose StringIterable

/**
 * StringIterable is the interface which defines the contract
 * the implementing StringIterable classes (Set, Map, List)
 * should keep to.
 */

StringIterable: interface
  iterator: method = .StringIterator
```

which returns an object implementing the `Iterator` interface. Every time `iterator()` is called, a new iterator is created, allowing multiple independent traversals of the same collection.

It enables code that obtains an iterator and repeatedly calls its `hasNext()` and `next()` methods.

This design illustrates one of the central principles of object-oriented programming: clients should depend upon capabilities rather than concrete implementations. The enhanced `for` loop does not require an `ArrayList`, a `LinkedList`, or a `HashSet`. It requires only something that can be iterated.

One of the strengths of `Iterable` is that it allows iteration over objects that are not traditional collections. A directory of files, the lines of a text file, the nodes of a syntax tree, the neighbours of a graph vertex, or the results of a database query can all naturally be represented as iterable objects. In such cases, the object does not necessarily *contain* its elements permanently; it merely provides a mechanism for visiting them one after another.

Implementing `Iterable` is often straightforward. A class need only provide an implementation of the `iterator()` method that returns an appropriate iterator. This iterator encapsulates whatever knowledge is necessary to traverse the underlying data structure. The collection itself remains responsible for storing its data, while the iterator becomes responsible for navigating through it. This separation of responsibilities is one of the reasons the iterator pattern has proven so successful.

From a design perspective, `Iterable` occupies an important position in the hierarchy of abstractions. It is intentionally minimal, expressing only the capability of sequential traversal. More specialized interfaces such as `List` and `Set` build upon this foundation by adding operations for insertion, removal, searching, and ordering.

In practice, a class should implement `Iterable` whenever its contents have a natural sequential traversal.

Notice that `Map` is deliberately separate. A map is not a collection of elements but a collection of associations between keys and values. Nevertheless, the collections returned by `keySet()`, `values()`, and `entrySet()` all implement `Iterable`, allowing their contents to be traversed using exactly the same mechanisms as any other collection.

4.16 The Iterator Interface

The `Iterator` interface provides a uniform mechanism for traversing the elements of a collection one at a time. It embodies the **Iterator** design pattern, one of the behavioural design patterns described by the "Gang of Four[3]," whose central purpose is to allow sequential access to the contents of an aggregate object without exposing its internal representation.

An iterator acts as a cursor positioned between the elements of a collection. Initially, the cursor is located before the first element. Repeated calls to `next()` advance the cursor through the collection, returning each element in turn until the end of the sequence is reached. This model allows clients to process arbitrarily large collections without requiring random access or exposing the collection's internal data structures.

The interface itself is deliberately compact. Its two essential methods are `hasNext()`, which determines whether another element is available, and `next()`, which returns that element while advancing the iterator to the next position. Together these methods define the basic protocol for sequential traversal.

One of the principal advantages of the iterator abstraction is that it completely decouples traversal from storage. The client need not know whether the underlying collection is implemented as a dynamic array, a linked list, a balanced tree, a hash table, or some more specialized data structure. The iterator encapsulates all knowledge of how to move from one element to the next, allowing the collection to change its internal representation without affecting client code.

Each call to an `Iterable` object's `iterator()` method produces a new iterator with its own independent traversal state. Consequently, multiple iterators may traverse the same collection simultaneously, each maintaining its own current position. One iterator may have reached the middle of the collection while another has only just begun, without interfering with one another. This independence is an important property of the design and enables many algorithms that require multiple concurrent traversals.

The order in which an iterator visits elements depends entirely upon the semantics of the underlying collection. A list iterator visits elements according to their positional order, a `LinkedHashSet` iterator follows insertion order, and a `TreeSet`

[3] Gamma et al. 1993

iterator produces elements in sorted order. By contrast, the iteration order of a `HashSet` or `HashMap` is intentionally unspecified. The iterator therefore reflects the logical ordering defined by the collection rather than imposing one of its own.

Because iteration is encapsulated in a separate object, collections are free to provide specialized iterators when appropriate. The `List` interface, for example, defines the richer `ListIterator` interface, which extends `Iterator` with bidirectional traversal, indexed positioning, and the ability to insert or replace elements during iteration. Other specialized collections may likewise provide iterators with additional capabilities while remaining compatible with the basic `Iterator` abstraction.

In practice, iterators should be preferred whenever the objective is to process every element of a collection sequentially without requiring indexed access. They provide a uniform programming model that applies equally well to lists, sets, queues, trees, and many user-defined collections, allowing algorithms to be written independently of the particular data structure being traversed. This separation of traversal from representation is one of the key design principles underlying the CREXX class library.

4.16.1 Live and Snapshot Iterators

An iterator provides a sequential view of the elements contained in a collection. While the basic purpose of an iterator is always the same—to traverse the elements of a collection without exposing its internal representation—not all iterators view the collection in the same way. The principal distinction is between **live iterators**, which remain connected to the underlying collection, and **snapshot iterators**, which traverse an immutable copy of the collection as it existed when the iterator was created.

A **live iterator** reflects the current state of its underlying collection. As the collection changes, the iterator may observe those changes, depending on the semantics of the particular collection. The iterator therefore represents a moving window onto the collection rather than a fixed picture of it. This approach is memory-efficient because no copy of the collection is required, and it is well suited to situations where the collection is not modified during iteration.

The disadvantage of a live iterator is that structural modifications to the collection may invalidate the iterator. Consider a list containing the elements *A*, *B*, *C*, and *D*. If an iterator has just returned *B* and another operation removes *C*, the iterator must somehow determine where to continue. Likewise, if an element is inserted before the iterator's current position, should the new element be visited or skipped? Different answers lead to different traversal semantics, and some situations make it impossible to continue safely.

A **snapshot iterator** takes a fundamentally different approach. Rather than traversing the original collection directly, it traverses a private copy created when the iterator is constructed. Once the snapshot has been taken, subsequent modifications to the original collection have no effect on the iterator. The sequence of elements returned by the iterator is therefore stable and deterministic, regardless of any later insertions, removals, or updates.

The principal advantage of snapshot iteration is safety. Because the iterator operates on an immutable view, it cannot be invalidated by changes to the underlying collection. Multiple threads may modify the original collection freely while existing iterators continue to traverse their own independent snapshots. The cost, however, is that creating the snapshot requires additional time and memory proportional to the size of the collection. Furthermore, the iterator does not reflect recent updates, since it represents the collection exactly as it existed at the instant the snapshot was taken.

The distinction between live and snapshot iterators mirrors a broader distinction found in database systems. A live iterator resembles reading directly from a table while updates are occurring, whereas a snapshot iterator resembles reading from a transaction snapshot that presents a consistent view of the database at a particular instant. Both approaches are valuable, but each is appropriate under different circumstances.

The choice between these two approaches ultimately reflects a trade-off between **freshness** and **stability**. A live iterator observes the current collection but may be invalidated by structural changes. A snapshot iterator guarantees a consistent traversal but observes only the past. Understanding this distinction is essential when designing collections and selecting the most appropriate iteration strategy for a particular application.

The CREXX `.stemIterator` offers a choice of **live** and **snapshot**-based iterators.

4.17 The .rexx class as a string container

The `.rexx` class offers functionality that is on par with the Object Rexx and NetRexx object-oriented notation for strings. It returns `.rexx` objects so calls can be chained.

4.18 The .stem class implements Rexx stem variables

The `.stem` class is recast as a collection class, a content-addressable container of `.string` while keeping the traditional Classic Rexx (dot) and the Object Rexx and NetRexx bracket notations, and extended with some readily usable container methods like `size()` and `.iterator`.

4.19 Notes on performance

The `StringTreeMap` class is based on an AVL Tree for optimal performance. This is a balanced binary tree with guaranteed $OlogN$ performance for all operations. This class is written in crexx and its performance has been benchmarked against a *red-black tree* implementation in C, which is what most class libraries use. Its performance is nearly identical to that native implementation; the native TreeMap, which is faster than the JVM version, is available but requires more investment in the build process of an application[4]. Also, its availability cannot be guaranteed for every platform crexx runs on.

4.20 Bag collections

A `Bag` is a collection that can contain more than one occurrence of the same object. In computer science this type of collection is also commonly called a *multiset*. The term `Bag` emphasizes an important property of the collection: objects are placed in the collection without assigning them a particular position, and the same object may be placed in it more than once.

Consider a bag containing the following objects:

[4] See the chapter on building application software in the Programming Guide.

```
apple
apple
orange
apple
pear
```

The bag contains five objects, of which three are distinct. The object `apple` occurs three times, while `orange` and `pear` each occur once. The number of occurrences is part of the state of the bag. A bag containing three apples is therefore different from a bag containing two apples, even when all other objects are the same.

This distinguishes a bag from both a `List` and a `Set`. A list also permits duplicate objects, but associates each occurrence with a position. A set does not associate objects with positions, but permits each object to occur only once. A bag occupies the useful middle ground: duplicate objects are retained, but their positions are not significant.

4.21 Relationship to the collection interfaces

`Bag` is a sibling of `Set`, rather than a specialization of it. A simplified view of the collection hierarchy is:

At first sight it might appear reasonable for `Bag` to implement `Set`, because both abstractions are concerned primarily with membership rather than position. Their contracts are nevertheless incompatible. The defining property of a set is that an object can occur at most once. Adding an object which is already a member of a set does not create another occurrence.

A bag has the opposite behaviour. For example:

```
bag.add("apple")
bag.add("apple")
bag.add("pear")
```

creates a collection containing three objects. The count of `apple` is two and the count of `pear` is one. Performing the equivalent operations on a set would produce a collection of only two objects.

Making `Bag` implement `Set` would consequently violate the meaning of the `Set` interface. Code receiving a `Set` is entitled to assume that every member occurs at most once. A bag cannot satisfy that assumption while still providing its defining behaviour.

Both abstractions therefore implement `Collection` independently. This also allows algorithms that require only general collection operations to work with sets, bags and lists without depending on their different rules for duplicate objects.

4.22 Size and occurrence counts

For a bag, `size()` denotes the total number of occurrences in the collection, not merely the number of distinct objects. If a bag contains three apples, two oranges and one pear, its size is six.

The number of occurrences of a particular object is obtained with `count()`:

```
bag.count("apple")     -- 3
bag.count("orange")    -- 2
bag.count("pear")      -- 1
bag.count("banana")    -- 0
```

A count of zero means that the object is not present. Consequently, `contains(object)` can be understood as testing whether:

```
bag.count(object) > 0
```

This relationship illustrates one of the differences between bags and sets. For a set, the count of an object could only ever be zero or one, so a separate count operation would add little information. For a bag, the count is an essential property of every member.

It is useful to distinguish the *size* of a bag from its number of distinct objects. A bag containing one thousand occurrences of the same object has a size of one thousand, although it contains only one distinct object. This distinction also has consequences for the way a bag can be implemented efficiently.

4.23 Adding and removing objects

Adding an object to a bag adds one occurrence. If the object is not already present, its count becomes one. If it is present, its count is increased by one. Repeated calls to `add()` therefore accumulate occurrences:

```
bag.add("apple")
```

```
bag.add("apple")
bag.add("apple")
```

After these operations the bag contains three occurrences of `apple`.

Removing an object performs the corresponding operation in the other direction. `remove()` removes one occurrence rather than necessarily removing the object completely. If an object has a count greater than one, its count is decreased. When its last occurrence is removed, the object is no longer a member of the bag.

Suppose a bag contains:

```
apple     3
orange    1
```

After:

```
bag.remove("apple")
```

the bag still contains `apple`, but its count is now two. Removing `orange`, on the other hand, causes `orange` to disappear from the bag because its count becomes zero.

This behaviour is important when using the methods inherited from `Collection`. Although a bag supports the familiar collection operations, their effects must be interpreted in terms of occurrences.

4.24 Iteration

Iteration over a bag visits the occurrences represented by the collection. A bag containing three apples, two oranges and one pear therefore supplies six objects through its iterator:

```
apple
apple
apple
orange
orange
pear
```

The order shown here is only illustrative. A bag does not inherently define an ordering. A hash-based implementation may return the objects in an implementation-dependent order, while an ordered implementation may return them according to a key or comparator.

The important property is that iteration reflects multiplicity. If an object has a count of three, an ordinary collection iterator returns that object three times. This keeps iteration consistent with `size()`: the number of objects produced by a complete iteration is the size of the bag.

Sometimes an application is interested only in the distinct objects. Those objects naturally form a set. For example, the bag:

```
apple
apple
orange
apple
pear
orange
```

has the distinct members:

```
apple
orange
pear
```

A bag implementation can therefore provide an operation that exposes its distinct members as a `Set`. This does not make the bag itself a set; it is a set-valued view of one aspect of the bag.

4.25 Equality

The occurrence counts are part of the value of a bag and must consequently be taken into account when bags are compared. The order in which objects were added is not significant.

For example, these two bags are equal:

```
Bag 1                  Bag 2

apple                   orange
apple                   apple
orange                  apple
```

Both contain two occurrences of `apple` and one occurrence of `orange`.

By contrast, a bag containing two apples and one orange is not equal to a bag containing one apple and two oranges. The two bags have the same size and the same set of distinct objects, but their occurrence counts differ.

This is another reason why a bag should not be regarded as a set. Comparing only the distinct members would discard information that is an essential part of the bag.

4.26 Implementation

A bag does not need to store every occurrence as a separate entry. A hash-based implementation can instead maintain a mapping from each distinct object to its occurrence count. Conceptually, a bag containing three apples, two oranges and one pear can be represented as:

```
apple  -> 3
orange -> 2
pear   -> 1
```

Adding an existing object increments its stored count, while removing an occurrence decrements it. When the count reaches zero, the map entry itself is removed.

This representation has an important advantage when duplicate objects are common. Its storage requirement is determined primarily by the number of distinct objects rather than by the total size of the bag. A bag containing one million occurrences of the same object needs only one map entry together with its count.

When the underlying map is a hash map, `add()`, `remove()`, `contains()` and `count()` normally have constant average-time complexity. An ordered bag can instead be based on a tree map when predictable ordering of the distinct objects is required.

The map used for the implementation remains an internal detail. From the outside, the bag behaves as a collection of objects and occurrences rather than as a map from objects to integers. This distinction should also be preserved by facilities such as `Printable`: the external representation describes the contents of the bag, not the data structure used to maintain its counts.

4.27 Choosing between a Bag, Set and List

The choice between these collection types follows directly from the information that an application needs to preserve. A `Set` is appropriate when only membership matters and duplicate objects must be eliminated. A `List` is appropriate when occurrences may be duplicated and their positions or sequence are significant.

A `Bag` is appropriate when duplicate occurrences are significant but their positions are not. It answers not only the question *is this object present?* but also *how many times is this object present?*

This makes bags particularly useful for frequency tables, inventories, histograms, word counts and similar applications. In all of these cases the number of occurrences carries information, while storing an explicit position for every occurrence would add semantics that the application does not require.

5

Date and Time

Programs have several different reasons for working with dates and times. A program may need to record a calendar date, calculate the number of days between two events, determine the ISO week containing a date, or identify an instant that can be compared with an event recorded in another part of the world. Although these requirements are related, they do not all require the same representation.

The CREXX `time` package therefore distinguishes between a date and time as it appears on a clock and a date and time that also carries an offset from UTC. It also provides support for converting these values to and from ISO 8601 representations.

The package is built around three classes:

- `DateTime` represents a calendar date and a time of day;
- `OffsetDateTime` adds an offset from UTC to a `DateTime`; and
- `ISO8601` converts these values to and from ISO 8601 text.

This separation is useful even in programs that never deal with different time zones. It makes explicit whether a value such as 20:00 means simply "eight o'clock" or identifies an instant relative to UTC.

5.1 Representing a Date and Time

A `DateTime` combines a Gregorian calendar date with a time of day. It has microsecond precision but contains no UTC offset.

The simplest way to create one is:

```
now = .DateTime()
```

which represents the current local date and time.

Internally, `DateTime` builds on the date facilities already present in Rexx. The calendar part is stored as a REXX base date: the number of days from the REXX base date. The time is represented independently as the number of microseconds since midnight.

This representation has a useful property. Calendar arithmetic can be performed on the base date while arithmetic within a day can be performed on a single integer. Crossing midnight then consists of carrying whole days between the two values.

A particular date and time can consequently be constructed directly when these values are known:

```
base = date("BASE", "2026-08-24", "INTERNATIONAL") as .int
noon = .DateTime(base, 43200000000)
```

The resulting object represents noon on 24 August 2026.

Normally an application does not need to work with these internal values. `DateTime` provides access to the familiar year, month, day, hour, minute, second and microsecond fields, as well as useful calendar properties such as the ordinal day within the year.

For example:

```
say noon.year()
say noon.month()
say noon.day()
say noon.hour()
```

produces the calendar and clock components of the value.

The underlying base-date representation nevertheless remains available. This is particularly useful when interfacing with existing REXX programs and functions that already use `DATE("BASE")`.

5.2 Calendar Arithmetic

Date and time arithmetic contains a number of boundary cases that are easy to miss when dates are represented merely as formatted strings. `DateTime` performs arithmetic on its internal date and time representation instead.

For example, adding a microsecond to:

```
2026-08-24T23:59:59.999999
```

produces:

```
2026-08-25T00:00:00.000000
```

The reverse operation crosses the boundary in the other direction. This same mechanism allows seconds, minutes and hours to cross one or several calendar days without special treatment by the application.

Arithmetic can be performed in units ranging from microseconds to weeks. A negative amount moves backwards in time:

```
value.addHours(-3)
```

Calendar arithmetic requires slightly different rules. A year is not a fixed number of days, and a month is not a fixed number of seconds. Leap years in particular make an operation such as “one year later” different from adding 365 days.

`addYears()` therefore performs calendar arithmetic. If a date is 29 February and the destination year is not a leap year, the resulting date is 28 February.

The class also exposes the calendar properties needed for this sort of work. It can determine whether its year is a leap year, the number of days in its month or year, and its position within the year.

5.3 Comparing DateTime Values

Two `DateTime` objects can be ordered chronologically because their calendar and clock components form a single local timeline.

For example:

```
if first.isBefore(second) then
    say "first comes before second"
```

Values can also be compared for equality or using a three-way comparison. Equality requires both the date and the time of day to be equal.

These operations compare `DateTime` values as represented. When an application needs to compare values carrying UTC offsets, `OffsetDateTime` provides the necessary additional information.

5.4 ISO Week Dates

The ordinary calendar year is not the only useful way of locating a day in a year. Business and administrative applications often use ISO week dates.

ISO weeks begin on Monday. Week 1 is the week containing 4 January, which is equivalent to saying that it is the week containing the first Thursday of the year.

An important consequence is that the ISO week-numbering year does not always start on 1 January.

For example, 1 January 2016 was a Friday and belongs to:

```
2015-W53-5
```

It is calendar year 2016, but ISO week 53 of week-numbering year 2015.

The reverse can also occur. 31 December 2018 was a Monday and belongs to:

```
2019-W01-1
```

`DateTime` provides the ISO weekday, week number and week-numbering year, and can produce the complete ISO week-date representation. It can also determine whether an ISO week year contains 52 or 53 weeks.

Keeping the calendar year and week-numbering year as separate concepts avoids a common source of errors around New Year.

5.5 Textual Representation

The normal string representation of a `DateTime` is deliberately predictable:

```
2026-08-24T20:57:57.123456
```

The fractional part always contains six digits, corresponding to the microsecond precision of the class.

This is a useful representation for display and diagnostics, but parsing and formatting ISO 8601 is kept in a separate class. This prevents the basic `DateTime` class from accumulating rules concerned only with textual representations.

5.6 Adding a UTC Offset

A local date and time is sufficient for many applications. In other cases the value must identify an instant relative to UTC.

`OffsetDateTime` adds this information without changing the meaning or implementation of `DateTime`. It consists simply of a `DateTime` and an offset from UTC expressed in minutes.

For example, suppose `dt` represents 22:00 on 24 August 2026:

```
value = .OffsetDateTime(dt, 120)
```

The offset of 120 minutes means that the local value is two hours ahead of UTC. The complete value can therefore be written as:

```
2026-08-24T22:00:00+02:00
```

Negative offsets work in the same way. An offset of -330 represents UTC-05:30.

The local representation is retained. Constructing or parsing the value above does not silently turn 22:00 into 20:00. This is useful because the object continues to represent exactly the date, time and offset supplied by the application.

When UTC is required, the conversion is explicit:

```
utc = value.toUTC()
```

For the preceding example this produces the equivalent value:

```
2026-08-24T20:00:00Z
```

UTC conversion also handles changes of calendar day. Thus:

```
2026-08-24T01:00:00+02:00
```

becomes:

```
2026-08-23T23:00:00Z
```

The original value is left unchanged.

This gives applications two useful views of the same information: the local date and time at the specified offset, and the corresponding UTC value.

5.7 ISO 8601

ISO 8601 provides a standard textual notation for dates and times. The `ISO8601` class performs the conversion between those representations and the date and time classes.

The distinction between `DateTime` and `OffsetDateTime` is reflected in the parser. A representation without an offset is parsed as a `DateTime`; a representation containing `Z` or a numeric UTC offset is parsed as an `OffsetDateTime`.

5.7.1 Local Date and Time

A calendar date can be parsed on its own:

```
iso = .ISO8601()
value = iso.parse("2026-08-24")
```

Because no time was specified, the result represents midnight:

```
2026-08-24T00:00:00.000000
```

A complete local date and time can be written as:

```
2026-08-24T12:34:56
```

Fractional seconds are optional:

```
2026-08-24T12:34:56.1
2026-08-24T12:34:56.123
2026-08-24T12:34:56.123456
```

`DateTime` has microsecond precision, so fractional values are normalized to six digits. The examples above consequently represent 100000, 123000 and 123456 microseconds respectively.

Input is checked while it is parsed. Invalid separators, incomplete times, clock fields outside their permitted ranges and fractions beyond the supported precision cause the `INVALID_ARGUMENTS` signal.

Applications that merely need to test input can use `isValid()`. It uses the same parser and handles `INVALID_ARGUMENTS` internally, so validation and parsing cannot gradually acquire different interpretations of the accepted syntax.

5.7.2 UTC and Numeric Offsets

An ISO 8601 date and time can identify UTC directly with `Z`:

```
2026-08-24T20:00:00Z
```

or specify a numeric offset:

```
2026-08-24T22:00:00+02:00
2026-08-24T14:30:00-05:30
```

These forms are parsed with `parseOffset()` and produce an `OffsetDateTime`.

The numeric representation is converted internally to an offset in minutes, so the examples correspond to offsets of 120 and -330 minutes. `Z` corresponds to an offset of zero.

Fractional seconds and offsets can be combined:

```
2026-08-24T22:00:00.123456+02:00
```

The parser preserves both the local time and the offset. Conversion to UTC, when wanted, remains an explicit operation on the resulting `OffsetDateTime`.

5.7.3 Formatting and Round Trips

The formatter performs the reverse conversion. A `DateTime` can be formatted as a calendar date, a date and time to second precision, or a date and time including microseconds.

An `OffsetDateTime` additionally includes its UTC offset. Zero is written in the conventional `Z` form, while other offsets are written numerically:

```
2026-08-24T20:00:00.000000Z
2026-08-24T22:00:00.123456+02:00
```

This allows parsed values to make a straightforward round trip:

```
text = "2026-08-24T22:00:00.123456+02:00"

value = iso.parseOffset(text)
result = iso.formatOffset(value)
```

`result` contains:

```
2026-08-24T22:00:00.123456+02:00
```

The parser and formatter therefore form the boundary between the textual ISO 8601 representation used for interchange and the `DateTime` and `OffsetDateTime` objects used by an application.

5.8 Choosing a Representation

The choice between the two date-time classes follows from the information available to the application.

Use a `DateTime` when the value consists of a calendar date and clock time:

```
2026-08-24T20:00:00
```

Use an `OffsetDateTime` when an explicit relationship with UTC is part of the value:

```
2026-08-24T20:00:00Z
2026-08-24T22:00:00+02:00
```

The `ISO8601` class preserves this distinction when values cross a textual interface. It does not invent an offset when none was supplied, and it does not discard one that was present.

This separation keeps the individual classes small while providing the building blocks needed for calendar calculations, ISO week dates, interchange formats and UTC-based comparisons.

6

All Classes

.stem rxfnsb

- .object
  - ∟ .stem *.–exposed*

*	public		**stem**	()
	public	**.int**	**hash**	(key=.string)
	public	**.string**	**get**	(key=.string)
	public	**.void**	**set**	(key="", value=.string)
	public	**.int**	**size**	()
	public	**.string**	**key**	(ix=.int)
	public	**.string**	**value**	(kv=.int)
	public	**.string**	**valueAt**	(ix=.int)
	public	**.string[]**	**tails**	()
	public	**.string[]**	**values**	()
	public	**.stemIterator**	**iterator**	()
	public	**.stemIterator**	**snapshotIterator**	()

.stemIterator rxfnsb

- .object
  - ∟ .stemIterator *.–exposed*

*	public		**stem**	()
	public	**.int**	**hash**	(key=.string)
	public	**.string**	**get**	(key=.string)
	public	**.void**	**set**	(key="", value=.string)
	public	**.int**	**size**	()
	public	**.string**	**key**	(ix=.int)
	public	**.string**	**value**	(kv=.int)
	public	**.string**	**valueAt**	(ix=.int)
	public	**.string[]**	**tails**	()
	public	**.string[]**	**values**	()
	public	**.stemIterator**	**iterator**	()
	public	**.stemIterator**	**snapshotIterator**	()
*	public		**stemIterator**	(source=reference .stem, snapshot_flag=0)
	public	**.int**	**hasNext**	()
	public	**.string**	**next**	()
	public	**.string**	**value**	()
	public	**.int**	**index**	()
	public	**.void**	**reset**	()
	public	**.int**	**isLive**	()

.jsondocument rxjson

- .object

⌞ .jsondocument *.–exposed*

*	public · · · · · · · · · · · ·		**jsondocument**	(json=.string)
	public · · · · · · · · · · · ·	**.int**	**valid**	()
	public · · · · · · · · · · · ·	**.string**	**error_code**	()
	public · · · · · · · · · · · ·	**.int**	**error_position**	()
	public · · · · · · · · · · · ·	**.string**	**error**	()
	public · · · · · · · · · · · ·	**.string**	**source**	()
	public · · · · · · · · · · · ·	**.int**	**source_bytes**	()
	public · · · · · · · · · · · ·	**.int**	**index_bytes**	()
	public · · · · · · · · · · · ·	**.int**	**node_count**	(node=0)
	public · · · · · · · · · · · ·	**.string**	**type**	(path=.string)
	public · · · · · · · · · · · ·	**.string**	**get**	(path=.string)
	public · · · · · · · · · · · ·	**.int**	**count**	(path=.string)
	public · · · · · · · · · · · ·	**.int**	**members**	(path=.string, expose names=.string[])
	public · · · · · · · · · · · ·	**.int**	**root**	()
	public · · · · · · · · · · · ·	**.int**	**find**	(path=.string)
	public · · · · · · · · · · · ·	**.int**	**member**	(parent=.int, name=.string)
	public · · · · · · · · · · · ·	**.int**	**element**	(parent=.int, index=.int)
	public · · · · · · · · · · · ·	**.string**	**node_type**	(node=.int)
	public · · · · · · · · · · · ·	**.string**	**node_get**	(node=.int)
	public · · · · · · · · · · · ·	**.int**	**children**	(node=.int, expose child_nodes=.int[])
	public · · · · · · · · · · · ·	**.string**	**node_name**	(node=.int)
	public · · · · · · · · · · · ·	**.int**	**node_string**	(node=.int, expose value=.string)
	public · · · · · · · · · · · ·	**.int**	**node_int**	(node=.int, expose value=.int)
	public · · · · · · · · · · · ·	**.int**	**node_float**	(node=.int, expose value=.float)
	public · · · · · · · · · · · ·	**.int**	**node_f32_array**	(node=.int, expected_count=.int, expose packed=.binary, expose error=.string)
	public · · · · · · · · · · · ·	**.int**	**node_i64_array**	(node=.int, expected_count=.int, expose packed=.binary, expose error=.string)
	public · · · · · · · · · · · ·	**.int**	**node_boolean**	(node=.int, expose value=.int)

.httpheaders rxfnsg

∘ .object

⌞ .httpheaders *.–exposed*

*	public		**empty**	()
*	public		**json**	()
*	public		**media_type**	(value=.string)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.httpheaders**	**add**	(name=.string, value=.string)
	public	**.channelvalue**	**to_channel**	()
	public	**.string**	**content_type**	()
	public	**.string**	**wire**	()
	public	**.int**	**count**	()
	public	**.int**	**bytes**	()
	public	**.string**	**header**	(name=.string)

.httppolicy

rxfnsg

- .object
  - .httppolicy

.–exposed

*	public		**empty**	()
*	public		**json**	()
*	public		**media_type**	(value=.string)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.httpheaders**	**add**	(name=.string, value=.string)
	public	**.channelvalue**	**to_channel**	()
	public	**.string**	**content_type**	()
	public	**.string**	**wire**	()
	public	**.int**	**count**	()
	public	**.int**	**bytes**	()
	public	**.string**	**header**	(name=.string)
*	public		**safe**	()
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.void**	**set_timeouts**	(dns=.int, connect=.int, tls=.int, request=.int, response=.int, total=.int)
	public	**.void**	**set_limits**	(header_count=.int, header_bytes=.int, request_bytes=.int)
	public	**.void**	**set_replay**	(redirects=.int, retries=.int)
	public	**.channelvalue**	**to_channel**	()
	public	**.void**	**validate**	()
	public	**.int**	**dns_timeout**	()
	public	**.int**	**connect_timeout**	()

public	.int	**tls_timeout**	()
public	.int	**request_timeout**	()
public	.int	**response_timeout**	()
public	.int	**total_timeout**	()
public	.int	**header_count**	()
public	.int	**header_bytes**	()
public	.int	**request_bytes**	()
public	.int	**redirects**	()
public	.int	**retries**	()

.httpbody rxfnsg

○ .object
  ∟ .httpbody *.–exposed*

*	public		**empty**	()
*	public		**json**	()
*	public		**media_type**	(value=.string)
*	public		**from_channel**	(encoded=.channelvalue)
	public	.httpheaders	**add**	(name=.string, value=.string)
	public	.channelvalue	**to_channel**	()
	public	.string	**content_type**	()
	public	.string	**wire**	()
	public	.int	**count**	()
	public	.int	**bytes**	()
	public	.string	**header**	(name=.string)
*	public		**safe**	()
*	public		**from_channel**	(encoded=.channelvalue)
	public	.void	**set_timeouts**	(dns=.int, connect=.int, tls=.int, request=.int, response=.int, total=.int)
	public	.void	**set_limits**	(header_count=.int, header_bytes=.int, request_bytes=.int)
	public	.void	**set_replay**	(redirects=.int, retries=.int)
	public	.channelvalue	**to_channel**	()
	public	.void	**validate**	()
	public	.int	**dns_timeout**	()
	public	.int	**connect_timeout**	()
	public	.int	**tls_timeout**	()
	public	.int	**request_timeout**	()
	public	.int	**response_timeout**	()

	public	.int	**total_timeout**	()
	public	.int	**header_count**	()
	public	.int	**header_bytes**	()
	public	.int	**request_bytes**	()
	public	.int	**redirects**	()
	public	.int	**retries**	()
*	public		**fixed**	(content_length=.int, capacity=.int, response_capacity=65536)
*	public		**chunked**	(capacity=.int, response_capacity=65536)
*	public		**from_channel**	(encoded=.channelvalue)
	public	.channelvalue	**to_channel**	()
	public	.int	**write**	(bytes=.binary, wait_milliseconds=.int)
	public	.void	**finish**	()
	public	.void	**cancel**	(reason=.string)
	public	.void	**close**	()
	public	.int	**length**	()
	public	.binary	**endpoint_reference**	()
	public	.binary	**response_reference**	()
	public	.int	**response_capacity**	()

.httpresponse rxfnsg

∘ .object
 ∟ .httpresponse *.–exposed*

*	public		**empty**	()
*	public		**json**	()
*	public		**media_type**	(value=.string)
*	public		**from_channel**	(encoded=.channelvalue)
	public	.httpheaders	**add**	(name=.string, value=.string)
	public	.channelvalue	**to_channel**	()
	public	.string	**content_type**	()
	public	.string	**wire**	()
	public	.int	**count**	()
	public	.int	**bytes**	()
	public	.string	**header**	(name=.string)
*	public		**safe**	()
*	public		**from_channel**	(encoded=.channelvalue)

	public	**.void**	**set_timeouts**	(dns=.int, connect=.int, tls=.int, request=.int, response=.int, total=.int)
	public	**.void**	**set_limits**	(header_count=.int, header_bytes=.int, request_bytes=.int)
	public	**.void**	**set_replay**	(redirects=.int, retries=.int)
	public	**.channelvalue**	**to_channel**	()
	public	**.void**	**validate**	()
	public	**.int**	**dns_timeout**	()
	public	**.int**	**connect_timeout**	()
	public	**.int**	**tls_timeout**	()
	public	**.int**	**request_timeout**	()
	public	**.int**	**response_timeout**	()
	public	**.int**	**total_timeout**	()
	public	**.int**	**header_count**	()
	public	**.int**	**header_bytes**	()
	public	**.int**	**request_bytes**	()
	public	**.int**	**redirects**	()
	public	**.int**	**retries**	()
*	public		**fixed**	(content_length=.int, capacity=.int, response_capacity=65536)
*	public		**chunked**	(capacity=.int, response_capacity=65536)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.channelvalue**	**to_channel**	()
	public	**.int**	**write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**finish**	()
	public	**.void**	**cancel**	(reason=.string)
	public	**.void**	**close**	()
	public	**.int**	**length**	()
	public	**.binary**	**endpoint_reference**	()
	public	**.binary**	**response_reference**	()
	public	**.int**	**response_capacity**	()

*	public		**new**	(transport_status=.int, http_status=.int, headers=.string, body=.string, error=.string, attempts=0, redirects=0, ambiguous=0)
*	public		**text**	(status=.int, body=.string, ?headers=.httpheaders)
*	public		**binary**	(status=.int, body=.binary, ?headers=.httpheaders)
*	public		**from_wire**	(wire=.binary)
*	public		**from_stream_wire**	(wire=.binary, stream_encoded=.binary)
	public	**.void**	**attach_stream**	(stream_encoded=.binary)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.channelvalue**	**to_channel**	()
	public	**.int**	**status**	()
	public	**.int**	**transport_status**	()
	public	**.boolean**	**succeeded**	()
	public	**.string**	**body**	()
	public	**.binary**	**body_bytes**	()
	public	**.string**	**raw**	()
	public	**.boolean**	**streaming**	()
	public	**.binary**	**body_reference**	()
	public	**.string**	**header**	(name=.string)
	public	**.string**	**header_wire**	()
	public	**.string**	**error**	()
	public	**.int**	**attempts**	()
	public	**.int**	**redirects**	()
	public	**.boolean**	**ambiguous_outcome**	()

._httpclient_contract rxfnsg

○ *._httpclient_contract* *.–exposed*

*	public		**empty**	()
*	public		**json**	()
*	public		**media_type**	(value=.string)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.httpheaders**	**add**	(name=.string, value=.string)
	public	**.channelvalue**	**to_channel**	()
	public	**.string**	**content_type**	()

	public	.string	**wire**	()
	public	.int	**count**	()
	public	.int	**bytes**	()
	public	.string	**header**	(name=.string)
*	public		**safe**	()
*	public		**from_channel**	(encoded=.channelvalue)
	public	.void	**set_timeouts**	(dns=.int, connect=.int, tls=.int, request=.int, response=.int, total=.int)
	public	.void	**set_limits**	(header_count=.int, header_bytes=.int, request_bytes=.int)
	public	.void	**set_replay**	(redirects=.int, retries=.int)
	public	.channelvalue	**to_channel**	()
	public	.void	**validate**	()
	public	.int	**dns_timeout**	()
	public	.int	**connect_timeout**	()
	public	.int	**tls_timeout**	()
	public	.int	**request_timeout**	()
	public	.int	**response_timeout**	()
	public	.int	**total_timeout**	()
	public	.int	**header_count**	()
	public	.int	**header_bytes**	()
	public	.int	**request_bytes**	()
	public	.int	**redirects**	()
	public	.int	**retries**	()
*	public		**fixed**	(content_length=.int, capacity=.int, response_capacity=65536)
*	public		**chunked**	(capacity=.int, response_capacity=65536)
*	public		**from_channel**	(encoded=.channelvalue)
	public	.channelvalue	**to_channel**	()
	public	.int	**write**	(bytes=.binary, wait_milliseconds=.int)
	public	.void	**finish**	()
	public	.void	**cancel**	(reason=.string)
	public	.void	**close**	()
	public	.int	**length**	()
	public	.binary	**endpoint_reference**	()
	public	.binary	**response_reference**	()
	public	.int	**response_capacity**	()

*	public		**new**	(transport_status=.int, http_status=.int, headers=.string, body=.string, error=.string, attempts=0, redirects=0, ambiguous=0)
*	public		**text**	(status=.int, body=.string, ?headers=.httpheaders)
*	public		**binary**	(status=.int, body=.binary, ?headers=.httpheaders)
*	public		**from_wire**	(wire=.binary)
*	public		**from_stream_wire**	(wire=.binary, stream_encoded=.binary)
	public	**.void**	**attach_stream**	(stream_encoded=.binary)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.channelvalue**	**to_channel**	()
	public	**.int**	**status**	()
	public	**.int**	**transport_status**	()
	public	**.boolean**	**succeeded**	()
	public	**.string**	**body**	()
	public	**.binary**	**body_bytes**	()
	public	**.string**	**raw**	()
	public	**.boolean**	**streaming**	()
	public	**.binary**	**body_reference**	()
	public	**.string**	**header**	(name=.string)
	public	**.string**	**header_wire**	()
	public	**.string**	**error**	()
	public	**.int**	**attempts**	()
	public	**.int**	**redirects**	()
	public	**.boolean**	**ambiguous_outcome**	()
*	public		**pooled**	(origin=.string, connections=.int, admission=.int, maximum_response=.int, ?policy=.httppolicy)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.channelvalue**	**to_channel**	()
	public	**.void**	**close**	()

.httpclient rxfnsg

◦ .object

*	public		**empty**	()
*	public		**json**	()
*	public		**media_type**	(value=.string)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.httpheaders**	**add**	(name=.string, value=.string)
	public	**.channelvalue**	**to_channel**	()
	public	**.string**	**content_type**	()
	public	**.string**	**wire**	()
	public	**.int**	**count**	()
	public	**.int**	**bytes**	()
	public	**.string**	**header**	(name=.string)
*	public		**safe**	()
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.void**	**set_timeouts**	(dns=.int, connect=.int, tls=.int, request=.int, response=.int, total=.int)
	public	**.void**	**set_limits**	(header_count=.int, header_bytes=.int, request_bytes=.int)
	public	**.void**	**set_replay**	(redirects=.int, retries=.int)
	public	**.channelvalue**	**to_channel**	()
	public	**.void**	**validate**	()
	public	**.int**	**dns_timeout**	()
	public	**.int**	**connect_timeout**	()
	public	**.int**	**tls_timeout**	()
	public	**.int**	**request_timeout**	()
	public	**.int**	**response_timeout**	()
	public	**.int**	**total_timeout**	()
	public	**.int**	**header_count**	()
	public	**.int**	**header_bytes**	()
	public	**.int**	**request_bytes**	()
	public	**.int**	**redirects**	()
	public	**.int**	**retries**	()
*	public		**fixed**	(content_length=.int, capacity=.int, response_capacity=65536)
*	public		**chunked**	(capacity=.int, response_capacity=65536)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.channelvalue**	**to_channel**	()

	public	.int	**write**	(bytes=.binary, wait_milliseconds=.int)
	public	.void	**finish**	()
	public	.void	**cancel**	(reason=.string)
	public	.void	**close**	()
	public	.int	**length**	()
	public	.binary	**endpoint_reference**	()
	public	.binary	**response_reference**	()
	public	.int	**response_capacity**	()
*	public		**new**	(transport_status=.int, http_status=.int, headers=.string, body=.string, error=.string, attempts=0, redirects=0, ambiguous=0)
*	public		**text**	(status=.int, body=.string, ?headers=.httpheaders)
*	public		**binary**	(status=.int, body=.binary, ?headers=.httpheaders)
*	public		**from_wire**	(wire=.binary)
*	public		**from_stream_wire**	(wire=.binary, stream_encoded=.binary)
	public	.void	**attach_stream**	(stream_encoded=.binary)
*	public		**from_channel**	(encoded=.channelvalue)
	public	.channelvalue	**to_channel**	()
	public	.int	**status**	()
	public	.int	**transport_status**	()
	public	.boolean	**succeeded**	()
	public	.string	**body**	()
	public	.binary	**body_bytes**	()
	public	.string	**raw**	()
	public	.boolean	**streaming**	()
	public	.binary	**body_reference**	()
	public	.string	**header**	(name=.string)
	public	.string	**header_wire**	()
	public	.string	**error**	()
	public	.int	**attempts**	()
	public	.int	**redirects**	()
	public	.boolean	**ambiguous_outcome**	()

*	public		**pooled**	(origin=.string, connections=.int, admission=.int, maximum_response=.int, ?policy=.httppolicy)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.channelvalue**	**to_channel**	()
	public	**.void**	**close**	()
*	public		**pooled**	(origin=.string, connections=.int, admission=.int, maximum_response=.int, ?policy=.httppolicy)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.channelvalue**	**to_channel**	()
	public	**.httpresponse**	**_request**	(verb=.string, path=.string, body=.binary, headers=.httpheaders)
	public	**.void**	**close**	()

.httpconnectionowner

rxfnsg

- .object
  - ∟ .httpconnectionowner

.–exposed

*	public		**empty**	()
*	public		**json**	()
*	public		**media_type**	(value=.string)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.httpheaders**	**add**	(name=.string, value=.string)
	public	**.channelvalue**	**to_channel**	()
	public	**.string**	**content_type**	()
	public	**.string**	**wire**	()
	public	**.int**	**count**	()
	public	**.int**	**bytes**	()
	public	**.string**	**header**	(name=.string)
*	public		**safe**	()
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.void**	**set_timeouts**	(dns=.int, connect=.int, tls=.int, request=.int, response=.int, total=.int)

	public	.void	**set_limits**	(header_count=.int, header_bytes=.int, request_bytes=.int)
	public	.void	**set_replay**	(redirects=.int, retries=.int)
	public	.channelvalue	**to_channel**	()
	public	.void	**validate**	()
	public	.int	**dns_timeout**	()
	public	.int	**connect_timeout**	()
	public	.int	**tls_timeout**	()
	public	.int	**request_timeout**	()
	public	.int	**response_timeout**	()
	public	.int	**total_timeout**	()
	public	.int	**header_count**	()
	public	.int	**header_bytes**	()
	public	.int	**request_bytes**	()
	public	.int	**redirects**	()
	public	.int	**retries**	()
*	public		**fixed**	(content_length=.int, capacity=.int, response_capacity=65536)
*	public		**chunked**	(capacity=.int, response_capacity=65536)
*	public		**from_channel**	(encoded=.channelvalue)
	public	.channelvalue	**to_channel**	()
	public	.int	**write**	(bytes=.binary, wait_milliseconds=.int)
	public	.void	**finish**	()
	public	.void	**cancel**	(reason=.string)
	public	.void	**close**	()
	public	.int	**length**	()
	public	.binary	**endpoint_reference**	()
	public	.binary	**response_reference**	()
	public	.int	**response_capacity**	()
*	public		**new**	(transport_status=.int, http_status=.int, headers=.string, body=.string, error=.string, attempts=0, redirects=0, ambiguous=0)
*	public		**text**	(status=.int, body=.string, ?headers=.httpheaders)

*	public		**binary**	(status=.int, body=.binary, ?headers=.httpheaders)
*	public		**from_wire**	(wire=.binary)
*	public		**from_stream_wire**	(wire=.binary, stream_encoded=.binary)
	public	**.void**	**attach_stream**	(stream_encoded=.binary)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.channelvalue**	**to_channel**	()
	public	**.int**	**status**	()
	public	**.int**	**transport_status**	()
	public	**.boolean**	**succeeded**	()
	public	**.string**	**body**	()
	public	**.binary**	**body_bytes**	()
	public	**.string**	**raw**	()
	public	**.boolean**	**streaming**	()
	public	**.binary**	**body_reference**	()
	public	**.string**	**header**	(name=.string)
	public	**.string**	**header_wire**	()
	public	**.string**	**error**	()
	public	**.int**	**attempts**	()
	public	**.int**	**redirects**	()
	public	**.boolean**	**ambiguous_outcome**	()
*	public		**pooled**	(origin=.string, connections=.int, admission=.int, maximum_response=.int, ?policy=.httppolicy)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.channelvalue**	**to_channel**	()
	public	**.void**	**close**	()
*	public		**pooled**	(origin=.string, connections=.int, admission=.int, maximum_response=.int, ?policy=.httppolicy)
*	public		**from_channel**	(encoded=.channelvalue)
	public	**.channelvalue**	**to_channel**	()
	public	**.httpresponse**	**_request**	(verb=.string, path=.string, body=.binary, headers=.httpheaders)
	public	**.void**	**close**	()

*	public		**httpconnectionowner**	(origin=.string, host=.string, port=.int, tls=.int, maximum_response=.int, dns_timeout=.int, connect_timeout=.int, tls_timeout=.int, request_timeout=.int, response_timeout=.int, header_count=.int, header_bytes=.int, request_bytes=.int, redirects=.int, retries=.int, ingress_reference=.channelvalue)
	public	**.channelvalue**	**run**	(ignored=.channelvalue, context=.taskcontext)
	public	**.binary**	**perform**	(wire=.binary)
	public	**.boolean**	**send_all**	(bytes=.binary)
	public	**.void**	**stream_failure**	(endpoint=.byteendpoint, transport=.int, status=.int, headers=.string, message=.string, attempts=.int, ambiguous=.boolean)
	public	**.boolean**	**receive_pending**	()
	public	**.string**	**read_stream_head**	()
	public	**.string**	**read_chunk_line**	()
	public	**.boolean**	**stream_fixed_response**	(endpoint=.byteendpoint, expected=.int)
	public	**.boolean**	**stream_chunked_response**	(endpoint=.byteendpoint)
	public	**.boolean**	**stream_close_response**	(endpoint=.byteendpoint)
	public	**.boolean**	**send_request_stream**	(endpoint=.byteendpoint, declared_length=.int)
	public	**.void**	**perform_stream**	(wire=.binary, response_endpoint=.byteendpoint)
	public	**.int**	**connect_socket**	()
	public	**.binary**	**read_response**	()
	public	**.void**	**discard_socket**	()

.httprequest rxfnsg

- .object

∟ .httprequest			.–exposed
* public		**inbound**	(verb=.string, target=.string, headers=.string, body=.binary, remote=.string)
* public		**routed**	(verb=.string, target=.string, headers=.string, body=.binary, remote=.string, reply_reference=.binary)
* public		**from_channel**	(encoded=.channelvalue)
public	**.channelvalue**	**to_channel**	()
public	**.string**	**verb**	()
public	**.string**	**target**	()
public	**.string**	**body**	()
public	**.binary**	**body_bytes**	()
public	**.string**	**header**	(name=.string)
public	**.string**	**headers**	()
public	**.string**	**remote**	()
public	**.boolean**	**routed_reply**	()
public	**.binary**	**reply_reference**	()

.httpservice rxfnsg

○ *.httpservice*			*.–exposed*
* public		**inbound**	(verb=.string, target=.string, headers=.string, body=.binary, remote=.string)
* public		**routed**	(verb=.string, target=.string, headers=.string, body=.binary, remote=.string, reply_reference=.binary)
* public		**from_channel**	(encoded=.channelvalue)
public	**.channelvalue**	**to_channel**	()
public	**.string**	**verb**	()
public	**.string**	**target**	()
public	**.string**	**body**	()
public	**.binary**	**body_bytes**	()
public	**.string**	**header**	(name=.string)

public	.string	**headers**	()
public	.string	**remote**	()
public	.boolean	**routed_reply**	()
public	.binary	**reply_reference**	()
public	.httpresponse	**handle**	(request=.httprequest, context=.taskcontext)
public	.channelvalue	**dispatch**	(encoded=.channelvalue, context=.taskcontext)

.httpserver rxfnsg

- .object
  - .httpserver *.–exposed*

*	public		**inbound**	(verb=.string, target=.string, headers=.string, body=.binary, remote=.string)
*	public		**routed**	(verb=.string, target=.string, headers=.string, body=.binary, remote=.string, reply_reference=.binary)
*	public		**from_channel**	(encoded=.channelvalue)
	public	.channelvalue	**to_channel**	()
	public	.string	**verb**	()
	public	.string	**target**	()
	public	.string	**body**	()
	public	.binary	**body_bytes**	()
	public	.string	**header**	(name=.string)
	public	.string	**headers**	()
	public	.string	**remote**	()
	public	.boolean	**routed_reply**	()
	public	.binary	**reply_reference**	()
	public	.httpresponse	**handle**	(request=.httprequest, context=.taskcontext)
	public	.channelvalue	**dispatch**	(encoded=.channelvalue, context=.taskcontext)

*	public		**local**	(host=.string, port=.int, workers=4, admission=64, connections=64, maximum_header_bytes=65536, maximum_headers=100, maximum_body_bytes=8388608, handler_timeout=30000, request_timeout=30000)
	public	**.void**	**_close_slot**	(slot=.int)
	public	**.int**	**_first_free_slot**	()
	public	**.boolean**	**_complete_slot**	(slot=.int, stop_after=.int)
	public	**.boolean**	**_read_slot**	(slot=.int, target=.tasktarget, stop_after=.int)
	public	**.int**	**serve**	(target=.tasktarget, maximum_requests=-1)
	public	**.void**	**close**	()
	public	**.string**	**host**	()
	public	**.int**	**port**	()
	public	**.int**	**served**	()
	public	**.int**	**scans**	()
	public	**.int**	**idle_scans**	()

.taskpool — concurrency

○ *.taskpool* *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()

.taskscope — concurrency

○ *.taskscope* *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)

	public	.int	**provider_type**	()
	public	.int	**capacity**	()
	public	.int	**admission_capacity**	()
	public	.int	**queued**	()
	public	.int	**running**	()
	public	.void	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	.task	**submit**	(target=.tasktarget, request=.channelvalue)
	public	.task	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	.task	**ask**	(service=.serviceref, request=.channelvalue)
	public	.completion	**next**	(wait_milliseconds=.int)
	public	.completion[]	**join**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.void	**close**	()
	public	.void	**finish**	()
	public	.void	**abort**	(reason=.channelvalue)
	public	.int	**result_integer**	(child=.task)
	public	.string	**result_string**	(child=.task)
	public	.binary	**result_binary**	(child=.task)
	public	.channelvalue	**result_value**	(child=.task)

.task — concurrency

○ *.task* — *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	.int	**provider_type**	()
	public	.int	**capacity**	()
	public	.int	**admission_capacity**	()
	public	.int	**queued**	()
	public	.int	**running**	()
	public	.void	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	.task	**submit**	(target=.tasktarget, request=.channelvalue)

public	.task	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
public	.task	**ask**	(service=.serviceref, request=.channelvalue)
public	.completion	**next**	(wait_milliseconds=.int)
public	.completion[]	**join**	()
public	.void	**cancel**	(reason=.channelvalue)
public	.void	**close**	()
public	.void	**finish**	()
public	.void	**abort**	(reason=.channelvalue)
public	.int	**result_integer**	(child=.task)
public	.string	**result_string**	(child=.task)
public	.binary	**result_binary**	(child=.task)
public	.channelvalue	**result_value**	(child=.task)
public	.int	**identity**	()
public	.boolean	**terminal**	()
public	.void	**cancel**	(reason=.channelvalue)
public	.completion	**wait**	(wait_milliseconds=.int)
public	.completion	**completion**	()

.tasktarget

concurrency

○ *.tasktarget* *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	.int	**provider_type**	()
	public	.int	**capacity**	()
	public	.int	**admission_capacity**	()
	public	.int	**queued**	()
	public	.int	**running**	()
	public	.void	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	.task	**submit**	(target=.tasktarget, request=.channelvalue)
	public	.task	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	.task	**ask**	(service=.serviceref, request=.channelvalue)
	public	.completion	**next**	(wait_milliseconds=.int)
	public	.completion[]	**join**	()
	public	.void	**cancel**	(reason=.channelvalue)

	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()

.taskwork concurrency

○ *.taskwork* *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()

	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue**	**run**	(request=.channelvalue, context=.taskcontext)

.taskcontext — concurrency

○ *.taskcontext* — *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)

	public	.void	**close**	()
	public	.void	**finish**	()
	public	.void	**abort**	(reason=.channelvalue)
	public	.int	**result_integer**	(child=.task)
	public	.string	**result_string**	(child=.task)
	public	.binary	**result_binary**	(child=.task)
	public	.channelvalue	**result_value**	(child=.task)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	.string	**name**	()
	public	.string	**signature**	()
	public	.channelvalue	**run**	(request=.channelvalue, context=.taskcontext)
	public	.int	**timeout_remaining**	()
	public	.boolean	**cancellation_requested**	()
	public	.string	**trace_identity**	()
	public	.byteendpoint	**endpoint**	(provider_reference=.channelvalue)

.completion — concurrency

○ *.completion* *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	.int	**provider_type**	()
	public	.int	**capacity**	()
	public	.int	**admission_capacity**	()
	public	.int	**queued**	()
	public	.int	**running**	()
	public	.void	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	.task	**submit**	(target=.tasktarget, request=.channelvalue)
	public	.task	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)

	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue**	**run**	(request=.channelvalue, context=.taskcontext)
	public	**.int**	**timeout_remaining**	()
	public	**.boolean**	**cancellation_requested**	()
	public	**.string**	**trace_identity**	()
	public	**.byteendpoint**	**endpoint**	(provider_reference=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()

.channel concurrency

○ *.channel* *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)

	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue**	**run**	(request=.channelvalue, context=.taskcontext)
	public	**.int**	**timeout_remaining**	()
	public	**.boolean**	**cancellation_requested**	()
	public	**.string**	**trace_identity**	()
	public	**.byteendpoint**	**endpoint**	(provider_reference=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()

	public	.int	**state**	()
	public	.boolean	**succeeded**	()
	public	.boolean	**has_value**	()
	public	.channelvalue	**value**	()
	public	.int	**error_code**	()
	public	.string	**message**	()
	public	.channelvalue	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	.int	**provider_type**	()
	public	.int	**capabilities**	()
	public	.channelrequest	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.void	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	.void	**close**	(mode=.int)

.channelrequest — concurrency

○ *.channelrequest* *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	.int	**provider_type**	()
	public	.int	**capacity**	()
	public	.int	**admission_capacity**	()
	public	.int	**queued**	()
	public	.int	**running**	()
	public	.void	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	.task	**submit**	(target=.tasktarget, request=.channelvalue)
	public	.task	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	.task	**ask**	(service=.serviceref, request=.channelvalue)
	public	.completion	**next**	(wait_milliseconds=.int)
	public	.completion[]	**join**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.void	**close**	()

	public	.void	**finish**	()
	public	.void	**abort**	(reason=.channelvalue)
	public	.int	**result_integer**	(child=.task)
	public	.string	**result_string**	(child=.task)
	public	.binary	**result_binary**	(child=.task)
	public	.channelvalue	**result_value**	(child=.task)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	.string	**name**	()
	public	.string	**signature**	()
	public	.channelvalue	**run**	(request=.channelvalue, context=.taskcontext)
	public	.int	**timeout_remaining**	()
	public	.boolean	**cancellation_requested**	()
	public	.string	**trace_identity**	()
	public	.byteendpoint	**endpoint**	(provider_reference=.channelvalue)
	public	.boolean	**available**	()
	public	.int	**request_identity**	()
	public	.int	**sequence**	()
	public	.int	**state**	()
	public	.boolean	**succeeded**	()
	public	.boolean	**has_value**	()
	public	.channelvalue	**value**	()
	public	.int	**error_code**	()
	public	.string	**message**	()
	public	.channelvalue	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	.int	**provider_type**	()
	public	.int	**capabilities**	()
	public	.channelrequest	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.void	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	.void	**close**	(mode=.int)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()

public	.void	cancel	(reason=.channelvalue)
public	.completion	wait	(wait_milliseconds=.int)
public	.completion	completion	()
public	.void	release	()

.channelvalue

concurrency

∘ *.channelvalue* *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()

*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	.string	**name**	()
	public	.string	**signature**	()
	public	.channelvalue	**run**	(request=.channelvalue, context=.taskcontext)
	public	.int	**timeout_remaining**	()
	public	.boolean	**cancellation_requested**	()
	public	.string	**trace_identity**	()
	public	.byteendpoint	**endpoint**	(provider_reference=.channelvalue)
	public	.boolean	**available**	()
	public	.int	**request_identity**	()
	public	.int	**sequence**	()
	public	.int	**state**	()
	public	.boolean	**succeeded**	()
	public	.boolean	**has_value**	()
	public	.channelvalue	**value**	()
	public	.int	**error_code**	()
	public	.string	**message**	()
	public	.channelvalue	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	.int	**provider_type**	()
	public	.int	**capabilities**	()
	public	.channelrequest	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.void	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	.void	**close**	(mode=.int)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])

*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()

.channelcodec concurrency

∘ *.channelcodec* *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()

	public	.void	**abort**	(reason=.channelvalue)
	public	.int	**result_integer**	(child=.task)
	public	.string	**result_string**	(child=.task)
	public	.binary	**result_binary**	(child=.task)
	public	.channelvalue	**result_value**	(child=.task)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	.string	**name**	()
	public	.string	**signature**	()
	public	.channelvalue	**run**	(request=.channelvalue, context=.taskcontext)
	public	.int	**timeout_remaining**	()
	public	.boolean	**cancellation_requested**	()
	public	.string	**trace_identity**	()
	public	.byteendpoint	**endpoint**	(provider_reference=.channelvalue)
	public	.boolean	**available**	()
	public	.int	**request_identity**	()
	public	.int	**sequence**	()
	public	.int	**state**	()
	public	.boolean	**succeeded**	()
	public	.boolean	**has_value**	()
	public	.channelvalue	**value**	()
	public	.int	**error_code**	()
	public	.string	**message**	()
	public	.channelvalue	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	.int	**provider_type**	()
	public	.int	**capabilities**	()
	public	.channelrequest	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.void	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	.void	**close**	(mode=.int)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)

	public	.completion	wait	(wait_milliseconds=.int)
	public	.completion	completion	()
*	public		null_value	()
*	public		boolean_value	(item=.boolean)
*	public		integer_value	(item=.int)
*	public		float_value	(item=.float)
*	public		decimal_value	(item=.decimal)
*	public		string_value	(item=.string)
*	public		binary_value	(item=.binary)
*	public		array_value	(items=.channelvalue[])
*	public		record_value	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	.int	type_code	()
	public	.string	schema	()
	public	.int	version	()
	public	.int	count	()
	public	.channelvalue	item	(index=.int)
	public	.channelvalue	field	(name=.string)
	public	.boolean	as_boolean	()
	public	.int	as_integer	()
	public	.float	as_float	()
	public	.decimal	as_decimal	()
	public	.string	as_string	()
	public	.binary	as_binary	()
	public	.binary	encode	()
	public	.string	schema	()
	public	.int	version	()
	public	.channelvalue	encode	(item=.object)
	public	.object	decode	(item=.channelvalue)

.byteendpoint concurrency

○ *.byteendpoint* *.–exposed*

*	public		local	(workers=.int, admission_capacity=.int)
*	public		process	(workers=.int, admission_capacity=.int)
	public	.int	provider_type	()
	public	.int	capacity	()
	public	.int	admission_capacity	()
	public	.int	queued	()
	public	.int	running	()
	public	.void	close	()
*	public		default	()

*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue**	**run**	(request=.channelvalue, context=.taskcontext)
	public	**.int**	**timeout_remaining**	()
	public	**.boolean**	**cancellation_requested**	()
	public	**.string**	**trace_identity**	()
	public	**.byteendpoint**	**endpoint**	(provider_reference=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()

*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.channelvalue**	**encode**	(item=.object)
	public	**.object**	**decode**	(item=.channelvalue)

*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()

.serviceref concurrency

○ *.serviceref* *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)

	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	.string	**name**	()
	public	.string	**signature**	()
	public	.channelvalue	**run**	(request=.channelvalue, context=.taskcontext)
	public	.int	**timeout_remaining**	()
	public	.boolean	**cancellation_requested**	()
	public	.string	**trace_identity**	()
	public	.byteendpoint	**endpoint**	(provider_reference=.channelvalue)
	public	.boolean	**available**	()
	public	.int	**request_identity**	()
	public	.int	**sequence**	()
	public	.int	**state**	()
	public	.boolean	**succeeded**	()
	public	.boolean	**has_value**	()
	public	.channelvalue	**value**	()
	public	.int	**error_code**	()
	public	.string	**message**	()
	public	.channelvalue	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	.int	**provider_type**	()
	public	.int	**capabilities**	()
	public	.channelrequest	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.void	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	.void	**close**	(mode=.int)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)

*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.channelvalue**	**encode**	(item=.object)
	public	**.object**	**decode**	(item=.channelvalue)
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.string**	**service_type**	()
	public	**.channelvalue**	**provider_reference**	()

.transferbuffer concurrency

○ *.transferbuffer* *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)

*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue**	**run**	(request=.channelvalue, context=.taskcontext)
	public	**.int**	**timeout_remaining**	()
	public	**.boolean**	**cancellation_requested**	()
	public	**.string**	**trace_identity**	()
	public	**.byteendpoint**	**endpoint**	(provider_reference=.channelvalue)
	public	**.boolean**	**available**	()

	public	.int	**request_identity**	()
	public	.int	**sequence**	()
	public	.int	**state**	()
	public	.boolean	**succeeded**	()
	public	.boolean	**has_value**	()
	public	.channelvalue	**value**	()
	public	.int	**error_code**	()
	public	.string	**message**	()
	public	.channelvalue	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	.int	**provider_type**	()
	public	.int	**capabilities**	()
	public	.channelrequest	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.void	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	.void	**close**	(mode=.int)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	.int	**type_code**	()
	public	.string	**schema**	()
	public	.int	**version**	()
	public	.int	**count**	()
	public	.channelvalue	**item**	(index=.int)
	public	.channelvalue	**field**	(name=.string)
	public	.boolean	**as_boolean**	()
	public	.int	**as_integer**	()
	public	.float	**as_float**	()

	public	.decimal	as_decimal	()
	public	.string	as_string	()
	public	.binary	as_binary	()
	public	.binary	encode	()
	public	.string	schema	()
	public	.int	version	()
	public	.channelvalue	encode	(item=.object)
	public	.object	decode	(item=.channelvalue)
*	public		memory	(direction=.int, capacity=.int)
*	public		null_endpoint	(direction=.int)
*	public		from_reference	(provider_reference=.channelvalue)
*	public		from_encoded_reference	(encoded=.binary)
	public	.channelvalue	provider_reference	()
	public	.channelrequest	start_read	(maximum_bytes=.int, wait_milliseconds=.int)
	public	.channelrequest	start_write	(bytes=.binary, wait_milliseconds=.int)
	public	.void	half_close	(direction=.int)
	public	.void	cancel	(reason=.channelvalue)
	public	.void	close	()
	public	.string	service_type	()
	public	.channelvalue	provider_reference	()
*	public		allocate	(capacity=.int)
*	public		copy_from	(bytes=.binary)
	public	.int	size	()
	public	.int	capacity	()
	public	.boolean	mutable	()
	public	.void	write	(offset=.int, bytes=.binary)
	public	.binary	read	(offset=.int, length=.int)
	public	.channelvalue	move_value	()
	public	.channelvalue	seal	()

.channelvalue_impl

concurrency

○ .object
  ∟ .channelvalue_impl

.–exposed

*	public		local	(workers=.int, admission_capacity=.int)
*	public		process	(workers=.int, admission_capacity=.int)
	public	.int	provider_type	()
	public	.int	capacity	()
	public	.int	admission_capacity	()
	public	.int	queued	()

	public	.int	**running**	()
	public	.void	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	.task	**submit**	(target=.tasktarget, request=.channelvalue)
	public	.task	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	.task	**ask**	(service=.serviceref, request=.channelvalue)
	public	.completion	**next**	(wait_milliseconds=.int)
	public	.completion[]	**join**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.void	**close**	()
	public	.void	**finish**	()
	public	.void	**abort**	(reason=.channelvalue)
	public	.int	**result_integer**	(child=.task)
	public	.string	**result_string**	(child=.task)
	public	.binary	**result_binary**	(child=.task)
	public	.channelvalue	**result_value**	(child=.task)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	.string	**name**	()
	public	.string	**signature**	()
	public	.channelvalue	**run**	(request=.channelvalue, context=.taskcontext)
	public	.int	**timeout_remaining**	()
	public	.boolean	**cancellation_requested**	()
	public	.string	**trace_identity**	()
	public	.byteendpoint	**endpoint**	(provider_reference=.channelvalue)
	public	.boolean	**available**	()
	public	.int	**request_identity**	()
	public	.int	**sequence**	()
	public	.int	**state**	()
	public	.boolean	**succeeded**	()
	public	.boolean	**has_value**	()
	public	.channelvalue	**value**	()

	public	.int	**error_code**	()
	public	.string	**message**	()
	public	.channelvalue	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	.int	**provider_type**	()
	public	.int	**capabilities**	()
	public	.channelrequest	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.void	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	.void	**close**	(mode=.int)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	.int	**type_code**	()
	public	.string	**schema**	()
	public	.int	**version**	()
	public	.int	**count**	()
	public	.channelvalue	**item**	(index=.int)
	public	.channelvalue	**field**	(name=.string)
	public	.boolean	**as_boolean**	()
	public	.int	**as_integer**	()
	public	.float	**as_float**	()
	public	.decimal	**as_decimal**	()
	public	.string	**as_string**	()
	public	.binary	**as_binary**	()
	public	.binary	**encode**	()
	public	.string	**schema**	()
	public	.int	**version**	()

	public	.channelvalue	**encode**	(item=.object)
	public	.object	**decode**	(item=.channelvalue)
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	.channelvalue	**provider_reference**	()
	public	.channelrequest	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	.channelrequest	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	.void	**half_close**	(direction=.int)
	public	.void	**cancel**	(reason=.channelvalue)
	public	.void	**close**	()
	public	.string	**service_type**	()
	public	.channelvalue	**provider_reference**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	.int	**size**	()
	public	.int	**capacity**	()
	public	.boolean	**mutable**	()
	public	.void	**write**	(offset=.int, bytes=.binary)
	public	.binary	**read**	(offset=.int, length=.int)
	public	.channelvalue	**move_value**	()
	public	.channelvalue	**seal**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
*	public		**from_encoded**	(encoded=.binary)
*	public		**local_capability**	(capability=.int, expected_kind=.int, provider=.int)
	public	.int	**provider_reference_type**	()
	public	.int	**provider_reference_rights**	()
	public	.int	**provider_reference_scope**	()

	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()

.transferbuffer_impl — concurrency

- .object
  - ∟ .transferbuffer_impl *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)

	public	.binary	**result_binary**	(child=.task)
	public	.channelvalue	**result_value**	(child=.task)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	.string	**name**	()
	public	.string	**signature**	()
	public	.channelvalue	**run**	(request=.channelvalue, context=.taskcontext)
	public	.int	**timeout_remaining**	()
	public	.boolean	**cancellation_requested**	()
	public	.string	**trace_identity**	()
	public	.byteendpoint	**endpoint**	(provider_reference=.channelvalue)
	public	.boolean	**available**	()
	public	.int	**request_identity**	()
	public	.int	**sequence**	()
	public	.int	**state**	()
	public	.boolean	**succeeded**	()
	public	.boolean	**has_value**	()
	public	.channelvalue	**value**	()
	public	.int	**error_code**	()
	public	.string	**message**	()
	public	.channelvalue	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	.int	**provider_type**	()
	public	.int	**capabilities**	()
	public	.channelrequest	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.void	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	.void	**close**	(mode=.int)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**null_value**	()

*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.channelvalue**	**encode**	(item=.object)
	public	**.object**	**decode**	(item=.channelvalue)
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.string**	**service_type**	()
	public	**.channelvalue**	**provider_reference**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()

	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
*	public		**from_encoded**	(encoded=.binary)
*	public		**local_capability**	(capability=.int, expected_kind=.int, provider=.int)
	public	**.int**	**provider_reference_type**	()
	public	**.int**	**provider_reference_rights**	()
	public	**.int**	**provider_reference_scope**	()
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()

public	**.void**	**write**	(offset=.int, bytes=.binary)
public	**.binary**	**read**	(offset=.int, length=.int)
public	**.channelvalue**	**move_value**	()
public	**.channelvalue**	**seal**	()

.completion_impl

concurrency

○ .object
∟ .completion_impl *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()

*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue**	**run**	(request=.channelvalue, context=.taskcontext)
	public	**.int**	**timeout_remaining**	()
	public	**.boolean**	**cancellation_requested**	()
	public	**.string**	**trace_identity**	()
	public	**.byteendpoint**	**endpoint**	(provider_reference=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])

*	public · · · · · · · · · · · ·		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public · · · · · · · · · · · ·	**.int**	**type_code**	()
	public · · · · · · · · · · · ·	**.string**	**schema**	()
	public · · · · · · · · · · · ·	**.int**	**version**	()
	public · · · · · · · · · · · ·	**.int**	**count**	()
	public · · · · · · · · · · · ·	**.channelvalue**	**item**	(index=.int)
	public · · · · · · · · · · · ·	**.channelvalue**	**field**	(name=.string)
	public · · · · · · · · · · · ·	**.boolean**	**as_boolean**	()
	public · · · · · · · · · · · ·	**.int**	**as_integer**	()
	public · · · · · · · · · · · ·	**.float**	**as_float**	()
	public · · · · · · · · · · · ·	**.decimal**	**as_decimal**	()
	public · · · · · · · · · · · ·	**.string**	**as_string**	()
	public · · · · · · · · · · · ·	**.binary**	**as_binary**	()
	public · · · · · · · · · · · ·	**.binary**	**encode**	()
	public · · · · · · · · · · · ·	**.string**	**schema**	()
	public · · · · · · · · · · · ·	**.int**	**version**	()
	public · · · · · · · · · · · ·	**.channelvalue**	**encode**	(item=.object)
	public · · · · · · · · · · · ·	**.object**	**decode**	(item=.channelvalue)
*	public · · · · · · · · · · · ·		**memory**	(direction=.int, capacity=.int)
*	public · · · · · · · · · · · ·		**null_endpoint**	(direction=.int)
*	public · · · · · · · · · · · ·		**from_reference**	(provider_reference=.channelvalue)
*	public · · · · · · · · · · · ·		**from_encoded_reference**	(encoded=.binary)
	public · · · · · · · · · · · ·	**.channelvalue**	**provider_reference**	()
	public · · · · · · · · · · · ·	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public · · · · · · · · · · · ·	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public · · · · · · · · · · · ·	**.void**	**half_close**	(direction=.int)
	public · · · · · · · · · · · ·	**.void**	**cancel**	(reason=.channelvalue)
	public · · · · · · · · · · · ·	**.void**	**close**	()
	public · · · · · · · · · · · ·	**.string**	**service_type**	()
	public · · · · · · · · · · · ·	**.channelvalue**	**provider_reference**	()
*	public · · · · · · · · · · · ·		**allocate**	(capacity=.int)
*	public · · · · · · · · · · · ·		**copy_from**	(bytes=.binary)
	public · · · · · · · · · · · ·	**.int**	**size**	()
	public · · · · · · · · · · · ·	**.int**	**capacity**	()
	public · · · · · · · · · · · ·	**.boolean**	**mutable**	()
	public · · · · · · · · · · · ·	**.void**	**write**	(offset=.int, bytes=.binary)
	public · · · · · · · · · · · ·	**.binary**	**read**	(offset=.int, length=.int)
	public · · · · · · · · · · · ·	**.channelvalue**	**move_value**	()

	public	.channelvalue	**seal**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
*	public		**from_encoded**	(encoded=.binary)
*	public		**local_capability**	(capability=.int, expected_kind=.int, provider=.int)
	public	.int	**provider_reference_type**	()
	public	.int	**provider_reference_rights**	()
	public	.int	**provider_reference_scope**	()
	public	.int	**type_code**	()
	public	.string	**schema**	()
	public	.int	**version**	()
	public	.int	**count**	()
	public	.channelvalue	**item**	(index=.int)
	public	.channelvalue	**field**	(name=.string)
	public	.boolean	**as_boolean**	()
	public	.int	**as_integer**	()
	public	.float	**as_float**	()
	public	.decimal	**as_decimal**	()
	public	.string	**as_string**	()
	public	.binary	**as_binary**	()
	public	.binary	**encode**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	.int	**size**	()
	public	.int	**capacity**	()
	public	.boolean	**mutable**	()
	public	.void	**write**	(offset=.int, bytes=.binary)
	public	.binary	**read**	(offset=.int, length=.int)
	public	.channelvalue	**move_value**	()
	public	.channelvalue	**seal**	()
*	public		**none**	()

*	public		**from_encoded**	(encoded=.binary, identity=.int)
*	public		**synthetic**	(identity=.int, sequence=.int, state=.int, error_code=.int, message=.string, details=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()

.channelrequest_impl — concurrency

○ .object
  ∟ .channelrequest_impl *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)

	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue**	**run**	(request=.channelvalue, context=.taskcontext)
	public	**.int**	**timeout_remaining**	()
	public	**.boolean**	**cancellation_requested**	()
	public	**.string**	**trace_identity**	()
	public	**.byteendpoint**	**endpoint**	(provider_reference=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.int**	**identity**	()

	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.channelvalue**	**encode**	(item=.object)
	public	**.object**	**decode**	(item=.channelvalue)
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()

	public	.string	**service_type**	()
	public	.channelvalue	**provider_reference**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	.int	**size**	()
	public	.int	**capacity**	()
	public	.boolean	**mutable**	()
	public	.void	**write**	(offset=.int, bytes=.binary)
	public	.binary	**read**	(offset=.int, length=.int)
	public	.channelvalue	**move_value**	()
	public	.channelvalue	**seal**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
*	public		**from_encoded**	(encoded=.binary)
*	public		**local_capability**	(capability=.int, expected_kind=.int, provider=.int)
	public	.int	**provider_reference_type**	()
	public	.int	**provider_reference_rights**	()
	public	.int	**provider_reference_scope**	()
	public	.int	**type_code**	()
	public	.string	**schema**	()
	public	.int	**version**	()
	public	.int	**count**	()
	public	.channelvalue	**item**	(index=.int)
	public	.channelvalue	**field**	(name=.string)
	public	.boolean	**as_boolean**	()
	public	.int	**as_integer**	()
	public	.float	**as_float**	()
	public	.decimal	**as_decimal**	()
	public	.string	**as_string**	()
	public	.binary	**as_binary**	()
	public	.binary	**encode**	()
*	public		**allocate**	(capacity=.int)

*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**none**	()
*	public		**from_encoded**	(encoded=.binary, identity=.int)
*	public		**synthetic**	(identity=.int, sequence=.int, state=.int, error_code=.int, message=.string, details=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()
*	public		**new**	(owner=reference .channel_-impl, ticket=.int, identity=.int)
*	public		**rejected**	(owner=reference .channel_-impl, identity=.int, sequence=.int, status=.int)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
	public	**.int**	**raw_ticket**	()
	public	**.completion_impl**	**cached**	()
	public	**.void**	**set_completion**	(item=.completion_impl)

	public	.void	**release**	()

.channel_impl — concurrency

- .object
  - ∟ .channel_impl *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	.int	**provider_type**	()
	public	.int	**capacity**	()
	public	.int	**admission_capacity**	()
	public	.int	**queued**	()
	public	.int	**running**	()
	public	.void	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	.task	**submit**	(target=.tasktarget, request=.channelvalue)
	public	.task	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	.task	**ask**	(service=.serviceref, request=.channelvalue)
	public	.completion	**next**	(wait_milliseconds=.int)
	public	.completion[]	**join**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.void	**close**	()
	public	.void	**finish**	()
	public	.void	**abort**	(reason=.channelvalue)
	public	.int	**result_integer**	(child=.task)
	public	.string	**result_string**	(child=.task)
	public	.binary	**result_binary**	(child=.task)
	public	.channelvalue	**result_value**	(child=.task)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	.string	**name**	()

	public	**.string**	**signature**	()
	public	**.channelvalue**	**run**	(request=.channelvalue, context=.taskcontext)
	public	**.int**	**timeout_remaining**	()
	public	**.boolean**	**cancellation_requested**	()
	public	**.string**	**trace_identity**	()
	public	**.byteendpoint**	**endpoint**	(provider_reference=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	**.int**	**type_code**	()

	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.channelvalue**	**encode**	(item=.object)
	public	**.object**	**decode**	(item=.channelvalue)
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.string**	**service_type**	()
	public	**.channelvalue**	**provider_reference**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)

*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
*	public		**from_encoded**	(encoded=.binary)
*	public		**local_capability**	(capability=.int, expected_kind=.int, provider=.int)
	public	**.int**	**provider_reference_type**	()
	public	**.int**	**provider_reference_rights**	()
	public	**.int**	**provider_reference_scope**	()
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**none**	()
*	public		**from_encoded**	(encoded=.binary, identity=.int)

*	public		**synthetic**	(identity=.int, sequence=.int, state=.int, error_code=.int, message=.string, details=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()
*	public		**new**	(owner=reference .channel_-impl, ticket=.int, identity=.int)
*	public		**rejected**	(owner=reference .channel_-impl, identity=.int, sequence=.int, status=.int)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
	public	**.int**	**raw_ticket**	()
	public	**.completion_impl**	**cached**	()
	public	**.void**	**set_completion**	(item=.completion_impl)
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.channelrequest_impl**	**start_structured**	(envelope=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)

public	**.void**	**cancel_ticket**	(ticket=.int, reason=.channelvalue)
public	**.completion_impl**	**wait_for**	(ticket=.int, wait_milliseconds=.int)
public	**.channelvalue_impl**	**local_capability**	()
public	**.boolean**	**open_state**	()
public	**.int**	**pending**	()
public	**.void**	**poll**	()
public	**.void**	**cancel_pending**	(reason=.channelvalue)
public	**.void**	**cancel_and_account**	()

.byteendpoint_impl — concurrency

○ .object
  ∟ .byteendpoint_impl *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()

	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	.string	**name**	()
	public	.string	**signature**	()
	public	.channelvalue	**run**	(request=.channelvalue, context=.taskcontext)
	public	.int	**timeout_remaining**	()
	public	.boolean	**cancellation_requested**	()
	public	.string	**trace_identity**	()
	public	.byteendpoint	**endpoint**	(provider_reference=.channelvalue)
	public	.boolean	**available**	()
	public	.int	**request_identity**	()
	public	.int	**sequence**	()
	public	.int	**state**	()
	public	.boolean	**succeeded**	()
	public	.boolean	**has_value**	()
	public	.channelvalue	**value**	()
	public	.int	**error_code**	()
	public	.string	**message**	()
	public	.channelvalue	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	.int	**provider_type**	()
	public	.int	**capabilities**	()
	public	.channelrequest	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.void	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	.void	**close**	(mode=.int)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)

*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.channelvalue**	**encode**	(item=.object)
	public	**.object**	**decode**	(item=.channelvalue)
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.string**	**service_type**	()
	public	**.channelvalue**	**provider_reference**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()

	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
*	public		**from_encoded**	(encoded=.binary)
*	public		**local_capability**	(capability=.int, expected_kind=.int, provider=.int)
	public	**.int**	**provider_reference_type**	()
	public	**.int**	**provider_reference_rights**	()
	public	**.int**	**provider_reference_scope**	()
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)

	public	.binary	**read**	(offset=.int, length=.int)
	public	.channelvalue	**move_value**	()
	public	.channelvalue	**seal**	()
*	public		**none**	()
*	public		**from_encoded**	(encoded=.binary, identity=.int)
*	public		**synthetic**	(identity=.int, sequence=.int, state=.int, error_code=.int, message=.string, details=.channelvalue)
	public	.boolean	**available**	()
	public	.int	**request_identity**	()
	public	.int	**sequence**	()
	public	.int	**state**	()
	public	.boolean	**succeeded**	()
	public	.boolean	**has_value**	()
	public	.channelvalue	**value**	()
	public	.int	**error_code**	()
	public	.string	**message**	()
	public	.channelvalue	**details**	()
*	public		**new**	(owner=reference .channel_-impl, ticket=.int, identity=.int)
*	public		**rejected**	(owner=reference .channel_-impl, identity=.int, sequence=.int, status=.int)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
	public	.int	**raw_ticket**	()
	public	.completion_impl	**cached**	()
	public	.void	**set_completion**	(item=.completion_impl)
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	.int	**provider_type**	()
	public	.int	**capabilities**	()

	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.channelrequest_impl**	**start_structured**	(envelope=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.void**	**cancel_ticket**	(ticket=.int, reason=.channelvalue)
	public	**.completion_impl**	**wait_for**	(ticket=.int, wait_milliseconds=.int)
	public	**.channelvalue_impl**	**local_capability**	()
	public	**.boolean**	**open_state**	()
	public	**.int**	**pending**	()
	public	**.void**	**poll**	()
	public	**.void**	**cancel_pending**	(reason=.channelvalue)
	public	**.void**	**cancel_and_account**	()
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()

.tasktarget_impl

concurrency

- .object
  - ∟ .tasktarget_impl — *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)

*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue**	**run**	(request=.channelvalue, context=.taskcontext)
	public	**.int**	**timeout_remaining**	()
	public	**.boolean**	**cancellation_requested**	()
	public	**.string**	**trace_identity**	()
	public	**.byteendpoint**	**endpoint**	(provider_reference=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()

*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.channelvalue**	**encode**	(item=.object)
	public	**.object**	**decode**	(item=.channelvalue)

*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.string**	**service_type**	()
	public	**.channelvalue**	**provider_reference**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
*	public		**from_encoded**	(encoded=.binary)
*	public		**local_capability**	(capability=.int, expected_kind=.int, provider=.int)
	public	**.int**	**provider_reference_type**	()
	public	**.int**	**provider_reference_rights**	()
	public	**.int**	**provider_reference_scope**	()
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()

	public	.int	**version**	()
	public	.int	**count**	()
	public	.channelvalue	**item**	(index=.int)
	public	.channelvalue	**field**	(name=.string)
	public	.boolean	**as_boolean**	()
	public	.int	**as_integer**	()
	public	.float	**as_float**	()
	public	.decimal	**as_decimal**	()
	public	.string	**as_string**	()
	public	.binary	**as_binary**	()
	public	.binary	**encode**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	.int	**size**	()
	public	.int	**capacity**	()
	public	.boolean	**mutable**	()
	public	.void	**write**	(offset=.int, bytes=.binary)
	public	.binary	**read**	(offset=.int, length=.int)
	public	.channelvalue	**move_value**	()
	public	.channelvalue	**seal**	()
*	public		**none**	()
*	public		**from_encoded**	(encoded=.binary, identity=.int)
*	public		**synthetic**	(identity=.int, sequence=.int, state=.int, error_code=.int, message=.string, details=.channelvalue)
	public	.boolean	**available**	()
	public	.int	**request_identity**	()
	public	.int	**sequence**	()
	public	.int	**state**	()
	public	.boolean	**succeeded**	()
	public	.boolean	**has_value**	()
	public	.channelvalue	**value**	()
	public	.int	**error_code**	()
	public	.string	**message**	()
	public	.channelvalue	**details**	()
*	public		**new**	(owner=reference .channel_-impl, ticket=.int, identity=.int)

*	public		**rejected**	(owner=reference .channel_-impl, identity=.int, sequence=.int, status=.int)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
	public	**.int**	**raw_ticket**	()
	public	**.completion_impl**	**cached**	()
	public	**.void**	**set_completion**	(item=.completion_impl)
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.channelrequest_impl**	**start_structured**	(envelope=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.void**	**cancel_ticket**	(ticket=.int, reason=.channelvalue)
	public	**.completion_impl**	**wait_for**	(ticket=.int, wait_milliseconds=.int)
	public	**.channelvalue_impl**	**local_capability**	()
	public	**.boolean**	**open_state**	()
	public	**.int**	**pending**	()
	public	**.void**	**poll**	()
	public	**.void**	**cancel_pending**	(reason=.channelvalue)
	public	**.void**	**cancel_and_account**	()
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)

	public	.void	**close**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
*	public		**sealed**	(diagnostic_name=.string, signature=.string, kind=.int, image_digest=.binary, callable_id=.int, adapter_callable_id=.int, signature_digest=.binary, factory_arguments=.channelvalue[])
	public	.string	**name**	()
	public	.string	**signature**	()
	public	.channelvalue_impl	**descriptor**	()

.taskarguments

concurrency

○ .object
∟ .taskarguments *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	.int	**provider_type**	()
	public	.int	**capacity**	()
	public	.int	**admission_capacity**	()
	public	.int	**queued**	()
	public	.int	**running**	()
	public	.void	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	.task	**submit**	(target=.tasktarget, request=.channelvalue)
	public	.task	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	.task	**ask**	(service=.serviceref, request=.channelvalue)
	public	.completion	**next**	(wait_milliseconds=.int)
	public	.completion[]	**join**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.void	**close**	()
	public	.void	**finish**	()

	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue**	**run**	(request=.channelvalue, context=.taskcontext)
	public	**.int**	**timeout_remaining**	()
	public	**.boolean**	**cancellation_requested**	()
	public	**.string**	**trace_identity**	()
	public	**.byteendpoint**	**endpoint**	(provider_reference=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)

	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.channelvalue**	**encode**	(item=.object)
	public	**.object**	**decode**	(item=.channelvalue)
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.string**	**service_type**	()
	public	**.channelvalue**	**provider_reference**	()

*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
*	public		**from_encoded**	(encoded=.binary)
*	public		**local_capability**	(capability=.int, expected_kind=.int, provider=.int)
	public	**.int**	**provider_reference_type**	()
	public	**.int**	**provider_reference_rights**	()
	public	**.int**	**provider_reference_scope**	()
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()

	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**none**	()
*	public		**from_encoded**	(encoded=.binary, identity=.int)
*	public		**synthetic**	(identity=.int, sequence=.int, state=.int, error_code=.int, message=.string, details=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()
*	public		**new**	(owner=reference .channel_-impl, ticket=.int, identity=.int)
*	public		**rejected**	(owner=reference .channel_-impl, identity=.int, sequence=.int, status=.int)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
	public	**.int**	**raw_ticket**	()
	public	**.completion_impl**	**cached**	()
	public	**.void**	**set_completion**	(item=.completion_impl)

*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.channelrequest_impl**	**start_structured**	(envelope=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.void**	**cancel_ticket**	(ticket=.int, reason=.channelvalue)
	public	**.completion_impl**	**wait_for**	(ticket=.int, wait_milliseconds=.int)
	public	**.channelvalue_impl**	**local_capability**	()
	public	**.boolean**	**open_state**	()
	public	**.int**	**pending**	()
	public	**.void**	**poll**	()
	public	**.void**	**cancel_pending**	(reason=.channelvalue)
	public	**.void**	**cancel_and_account**	()
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])

*	public		**sealed**	(diagnostic_name=.string, signature=.string, kind=.int, image_digest=.binary, callable_id=.int, adapter_callable_id=.int, signature_digest=.binary, factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue_impl**	**descriptor**	()
*	public		**empty**	()
	public	**.taskarguments**	**add_value**	(value=.channelvalue)
	public	**.taskarguments**	**add_channel_value**	(value=.channelvalue)
	public	**.taskarguments**	**add_boolean**	(value=.boolean)
	public	**.taskarguments**	**add_integer**	(value=.int)
	public	**.taskarguments**	**add_string**	(value=.string)
	public	**.taskarguments**	**add_binary**	(value=.binary)
	public	**.channelvalue[]**	**values**	()

.task_impl

concurrency

∘ .object
 ∟ .task_impl *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()

	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue**	**run**	(request=.channelvalue, context=.taskcontext)
	public	**.int**	**timeout_remaining**	()
	public	**.boolean**	**cancellation_requested**	()
	public	**.string**	**trace_identity**	()
	public	**.byteendpoint**	**endpoint**	(provider_reference=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)

	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.channelvalue**	**encode**	(item=.object)
	public	**.object**	**decode**	(item=.channelvalue)
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)

	public	**.void**	**close**	()
	public	**.string**	**service_type**	()
	public	**.channelvalue**	**provider_reference**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
*	public		**from_encoded**	(encoded=.binary)
*	public		**local_capability**	(capability=.int, expected_kind=.int, provider=.int)
	public	**.int**	**provider_reference_type**	()
	public	**.int**	**provider_reference_rights**	()
	public	**.int**	**provider_reference_scope**	()
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()

*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**none**	()
*	public		**from_encoded**	(encoded=.binary, identity=.int)
*	public		**synthetic**	(identity=.int, sequence=.int, state=.int, error_code=.int, message=.string, details=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()
*	public		**new**	(owner=reference .channel_-impl, ticket=.int, identity=.int)
*	public		**rejected**	(owner=reference .channel_-impl, identity=.int, sequence=.int, status=.int)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
	public	**.int**	**raw_ticket**	()
	public	**.completion_impl**	**cached**	()

	public	**.void**	**set_completion**	(item=.completion_impl)
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.channelrequest_impl**	**start_structured**	(envelope=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.void**	**cancel_ticket**	(ticket=.int, reason=.channelvalue)
	public	**.completion_impl**	**wait_for**	(ticket=.int, wait_milliseconds=.int)
	public	**.channelvalue_impl**	**local_capability**	()
	public	**.boolean**	**open_state**	()
	public	**.int**	**pending**	()
	public	**.void**	**poll**	()
	public	**.void**	**cancel_pending**	(reason=.channelvalue)
	public	**.void**	**cancel_and_account**	()
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])

*	public		**sealed**	(diagnostic_name=.string, signature=.string, kind=.int, image_digest=.binary, callable_id=.int, adapter_callable_id=.int, signature_digest=.binary, factory_arguments=.channelvalue[])
	public	.string	**name**	()
	public	.string	**signature**	()
	public	.channelvalue_impl	**descriptor**	()
*	public		**empty**	()
	public	.taskarguments	**add_value**	(value=.channelvalue)
	public	.taskarguments	**add_channel_value**	(value=.channelvalue)
	public	.taskarguments	**add_boolean**	(value=.boolean)
	public	.taskarguments	**add_integer**	(value=.int)
	public	.taskarguments	**add_string**	(value=.string)
	public	.taskarguments	**add_binary**	(value=.binary)
	public	.channelvalue[]	**values**	()
*	public		**new**	(scope=reference .taskscope_-impl, channel=reference .channel_-impl, ticket=.int, identity=.int)
*	public		**rejected**	(scope=reference .taskscope_-impl, channel=reference .channel_-impl, identity=.int, cached=.completion_impl)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()

.taskpool_impl

concurrency

○ .object
  ∟ .taskpool_impl *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)

	public · · · · · · · · · · · ·	.int	**provider_type**	()
	public · · · · · · · · · · · ·	.int	**capacity**	()
	public · · · · · · · · · · · ·	.int	**admission_capacity**	()
	public · · · · · · · · · · · ·	.int	**queued**	()
	public · · · · · · · · · · · ·	.int	**running**	()
	public · · · · · · · · · · · ·	.void	**close**	()
*	public · · · · · · · · · · · ·		**default**	()
*	public · · · · · · · · · · · ·		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public · · · · · · · · · · · ·		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public · · · · · · · · · · · ·	.task	**submit**	(target=.tasktarget, request=.channelvalue)
	public · · · · · · · · · · · ·	.task	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public · · · · · · · · · · · ·	.task	**ask**	(service=.serviceref, request=.channelvalue)
	public · · · · · · · · · · · ·	.completion	**next**	(wait_milliseconds=.int)
	public · · · · · · · · · · · ·	.completion[]	**join**	()
	public · · · · · · · · · · · ·	.void	**cancel**	(reason=.channelvalue)
	public · · · · · · · · · · · ·	.void	**close**	()
	public · · · · · · · · · · · ·	.void	**finish**	()
	public · · · · · · · · · · · ·	.void	**abort**	(reason=.channelvalue)
	public · · · · · · · · · · · ·	.int	**result_integer**	(child=.task)
	public · · · · · · · · · · · ·	.string	**result_string**	(child=.task)
	public · · · · · · · · · · · ·	.binary	**result_binary**	(child=.task)
	public · · · · · · · · · · · ·	.channelvalue	**result_value**	(child=.task)
	public · · · · · · · · · · · ·	.int	**identity**	()
	public · · · · · · · · · · · ·	.boolean	**terminal**	()
	public · · · · · · · · · · · ·	.void	**cancel**	(reason=.channelvalue)
	public · · · · · · · · · · · ·	.completion	**wait**	(wait_milliseconds=.int)
	public · · · · · · · · · · · ·	.completion	**completion**	()
*	public · · · · · · · · · · · ·		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public · · · · · · · · · · · ·	.string	**name**	()
	public · · · · · · · · · · · ·	.string	**signature**	()
	public · · · · · · · · · · · ·	.channelvalue	**run**	(request=.channelvalue, context=.taskcontext)
	public · · · · · · · · · · · ·	.int	**timeout_remaining**	()
	public · · · · · · · · · · · ·	.boolean	**cancellation_requested**	()
	public · · · · · · · · · · · ·	.string	**trace_identity**	()
	public · · · · · · · · · · · ·	.byteendpoint	**endpoint**	(provider_reference=.channelvalue)
	public · · · · · · · · · · · ·	.boolean	**available**	()
	public · · · · · · · · · · · ·	.int	**request_identity**	()
	public · · · · · · · · · · · ·	.int	**sequence**	()

	public	.int	**state**	()
	public	.boolean	**succeeded**	()
	public	.boolean	**has_value**	()
	public	.channelvalue	**value**	()
	public	.int	**error_code**	()
	public	.string	**message**	()
	public	.channelvalue	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	.int	**provider_type**	()
	public	.int	**capabilities**	()
	public	.channelrequest	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.void	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	.void	**close**	(mode=.int)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	.int	**type_code**	()
	public	.string	**schema**	()
	public	.int	**version**	()
	public	.int	**count**	()
	public	.channelvalue	**item**	(index=.int)
	public	.channelvalue	**field**	(name=.string)
	public	.boolean	**as_boolean**	()
	public	.int	**as_integer**	()
	public	.float	**as_float**	()
	public	.decimal	**as_decimal**	()
	public	.string	**as_string**	()

	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.channelvalue**	**encode**	(item=.object)
	public	**.object**	**decode**	(item=.channelvalue)
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.string**	**service_type**	()
	public	**.channelvalue**	**provider_reference**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
*	public		**from_encoded**	(encoded=.binary)

*	public		**local_capability**	(capability=.int, expected_kind=.int, provider=.int)
	public	**.int**	**provider_reference_type**	()
	public	**.int**	**provider_reference_rights**	()
	public	**.int**	**provider_reference_scope**	()
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**none**	()
*	public		**from_encoded**	(encoded=.binary, identity=.int)
*	public		**synthetic**	(identity=.int, sequence=.int, state=.int, error_code=.int, message=.string, details=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()

	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()
*	public		**new**	(owner=reference .channel_-impl, ticket=.int, identity=.int)
*	public		**rejected**	(owner=reference .channel_-impl, identity=.int, sequence=.int, status=.int)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
	public	**.int**	**raw_ticket**	()
	public	**.completion_impl**	**cached**	()
	public	**.void**	**set_completion**	(item=.completion_impl)
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.channelrequest_impl**	**start_structured**	(envelope=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.void**	**cancel_ticket**	(ticket=.int, reason=.channelvalue)
	public	**.completion_impl**	**wait_for**	(ticket=.int, wait_milliseconds=.int)
	public	**.channelvalue_impl**	**local_capability**	()
	public	**.boolean**	**open_state**	()
	public	**.int**	**pending**	()
	public	**.void**	**poll**	()
	public	**.void**	**cancel_pending**	(reason=.channelvalue)
	public	**.void**	**cancel_and_account**	()
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)

*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
*	public		**sealed**	(diagnostic_name=.string, signature=.string, kind=.int, image_digest=.binary, callable_id=.int, adapter_callable_id=.int, signature_digest=.binary, factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue_impl**	**descriptor**	()
*	public		**empty**	()
	public	**.taskarguments**	**add_value**	(value=.channelvalue)
	public	**.taskarguments**	**add_channel_value**	(value=.channelvalue)
	public	**.taskarguments**	**add_boolean**	(value=.boolean)
	public	**.taskarguments**	**add_integer**	(value=.int)
	public	**.taskarguments**	**add_string**	(value=.string)
	public	**.taskarguments**	**add_binary**	(value=.binary)
	public	**.channelvalue[]**	**values**	()
*	public		**new**	(scope=reference .taskscope_- impl, channel=reference .channel_- impl, ticket=.int, identity=.int)
*	public		**rejected**	(scope=reference .taskscope_- impl, channel=reference .channel_- impl, identity=.int, cached=.completion_impl)
	public	**.int**	**identity**	()

	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
	public	**.channel_impl**	**pool_channel**	()
	public	**.boolean**	**open_state**	()

.taskscope_impl

concurrency

○ .object
  ∟ .taskscope_impl *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()

	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue**	**run**	(request=.channelvalue, context=.taskcontext)
	public	**.int**	**timeout_remaining**	()
	public	**.boolean**	**cancellation_requested**	()
	public	**.string**	**trace_identity**	()
	public	**.byteendpoint**	**endpoint**	(provider_reference=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)

	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.channelvalue**	**encode**	(item=.object)
	public	**.object**	**decode**	(item=.channelvalue)
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.string**	**service_type**	()
	public	**.channelvalue**	**provider_reference**	()

*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
*	public		**from_encoded**	(encoded=.binary)
*	public		**local_capability**	(capability=.int, expected_kind=.int, provider=.int)
	public	**.int**	**provider_reference_type**	()
	public	**.int**	**provider_reference_rights**	()
	public	**.int**	**provider_reference_scope**	()
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()

	public	.int	**capacity**	()
	public	.boolean	**mutable**	()
	public	.void	**write**	(offset=.int, bytes=.binary)
	public	.binary	**read**	(offset=.int, length=.int)
	public	.channelvalue	**move_value**	()
	public	.channelvalue	**seal**	()
*	public		**none**	()
*	public		**from_encoded**	(encoded=.binary, identity=.int)
*	public		**synthetic**	(identity=.int, sequence=.int, state=.int, error_code=.int, message=.string, details=.channelvalue)
	public	.boolean	**available**	()
	public	.int	**request_identity**	()
	public	.int	**sequence**	()
	public	.int	**state**	()
	public	.boolean	**succeeded**	()
	public	.boolean	**has_value**	()
	public	.channelvalue	**value**	()
	public	.int	**error_code**	()
	public	.string	**message**	()
	public	.channelvalue	**details**	()
*	public		**new**	(owner=reference .channel_-impl, ticket=.int, identity=.int)
*	public		**rejected**	(owner=reference .channel_-impl, identity=.int, sequence=.int, status=.int)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
	public	.int	**raw_ticket**	()
	public	.completion_impl	**cached**	()
	public	.void	**set_completion**	(item=.completion_impl)

*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.channelrequest_impl**	**start_structured**	(envelope=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.void**	**cancel_ticket**	(ticket=.int, reason=.channelvalue)
	public	**.completion_impl**	**wait_for**	(ticket=.int, wait_milliseconds=.int)
	public	**.channelvalue_impl**	**local_capability**	()
	public	**.boolean**	**open_state**	()
	public	**.int**	**pending**	()
	public	**.void**	**poll**	()
	public	**.void**	**cancel_pending**	(reason=.channelvalue)
	public	**.void**	**cancel_and_account**	()
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])

*	public		**sealed**	(diagnostic_name=.string, signature=.string, kind=.int, image_digest=.binary, callable_id=.int, adapter_callable_id=.int, signature_digest=.binary, factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue_impl**	**descriptor**	()
*	public		**empty**	()
	public	**.taskarguments**	**add_value**	(value=.channelvalue)
	public	**.taskarguments**	**add_channel_value**	(value=.channelvalue)
	public	**.taskarguments**	**add_boolean**	(value=.boolean)
	public	**.taskarguments**	**add_integer**	(value=.int)
	public	**.taskarguments**	**add_string**	(value=.string)
	public	**.taskarguments**	**add_binary**	(value=.binary)
	public	**.channelvalue[]**	**values**	()
*	public		**new**	(scope=reference .taskscope_- impl, channel=reference .channel_- impl, ticket=.int, identity=.int)
*	public		**rejected**	(scope=reference .taskscope_- impl, channel=reference .channel_- impl, identity=.int, cached=.completion_impl)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()

	public	**.void**	**close**	()
	public	**.channel_impl**	**pool_channel**	()
	public	**.boolean**	**open_state**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.completion_impl**	**wait_task**	(identity=.int, wait_milliseconds=.int)
	public	**.void**	**mark_observed**	(identity=.int)
	public	**.int**	**pending**	()
	public	**.boolean**	**open_state**	()

.taskcontext_impl

concurrency

○ .object
  ∟ .taskcontext_impl *.–exposed*

*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)
	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)

*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
	public	**.string**	**name**	()
	public	**.string**	**signature**	()
	public	**.channelvalue**	**run**	(request=.channelvalue, context=.taskcontext)
	public	**.int**	**timeout_remaining**	()
	public	**.boolean**	**cancellation_requested**	()
	public	**.string**	**trace_identity**	()
	public	**.byteendpoint**	**endpoint**	(provider_reference=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()

*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
	public	**.string**	**schema**	()
	public	**.int**	**version**	()
	public	**.channelvalue**	**encode**	(item=.object)
	public	**.object**	**decode**	(item=.channelvalue)

*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.string**	**service_type**	()
	public	**.channelvalue**	**provider_reference**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**null_value**	()
*	public		**boolean_value**	(item=.boolean)
*	public		**integer_value**	(item=.int)
*	public		**float_value**	(item=.float)
*	public		**decimal_value**	(item=.decimal)
*	public		**string_value**	(item=.string)
*	public		**binary_value**	(item=.binary)
*	public		**array_value**	(items=.channelvalue[])
*	public		**record_value**	(schema=.string, version=.int, names=.string[], values=.channelvalue[])
*	public		**from_encoded**	(encoded=.binary)
*	public		**local_capability**	(capability=.int, expected_kind=.int, provider=.int)
	public	**.int**	**provider_reference_type**	()
	public	**.int**	**provider_reference_rights**	()
	public	**.int**	**provider_reference_scope**	()
	public	**.int**	**type_code**	()
	public	**.string**	**schema**	()

	public	**.int**	**version**	()
	public	**.int**	**count**	()
	public	**.channelvalue**	**item**	(index=.int)
	public	**.channelvalue**	**field**	(name=.string)
	public	**.boolean**	**as_boolean**	()
	public	**.int**	**as_integer**	()
	public	**.float**	**as_float**	()
	public	**.decimal**	**as_decimal**	()
	public	**.string**	**as_string**	()
	public	**.binary**	**as_binary**	()
	public	**.binary**	**encode**	()
*	public		**allocate**	(capacity=.int)
*	public		**copy_from**	(bytes=.binary)
	public	**.int**	**size**	()
	public	**.int**	**capacity**	()
	public	**.boolean**	**mutable**	()
	public	**.void**	**write**	(offset=.int, bytes=.binary)
	public	**.binary**	**read**	(offset=.int, length=.int)
	public	**.channelvalue**	**move_value**	()
	public	**.channelvalue**	**seal**	()
*	public		**none**	()
*	public		**from_encoded**	(encoded=.binary, identity=.int)
*	public		**synthetic**	(identity=.int, sequence=.int, state=.int, error_code=.int, message=.string, details=.channelvalue)
	public	**.boolean**	**available**	()
	public	**.int**	**request_identity**	()
	public	**.int**	**sequence**	()
	public	**.int**	**state**	()
	public	**.boolean**	**succeeded**	()
	public	**.boolean**	**has_value**	()
	public	**.channelvalue**	**value**	()
	public	**.int**	**error_code**	()
	public	**.string**	**message**	()
	public	**.channelvalue**	**details**	()
*	public		**new**	(owner=reference .channel_-impl, ticket=.int, identity=.int)

*	public		**rejected**	(owner=reference .channel_-impl, identity=.int, sequence=.int, status=.int)
	public	**.int**	**identity**	()
	public	**.boolean**	**terminal**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.completion**	**completion**	()
	public	**.int**	**raw_ticket**	()
	public	**.completion_impl**	**cached**	()
	public	**.void**	**set_completion**	(item=.completion_impl)
*	public		**open**	(provider_type=.int, required_capabilities=.int, configuration=.channelvalue)
	public	**.int**	**provider_type**	()
	public	**.int**	**capabilities**	()
	public	**.channelrequest**	**start**	(envelope=.channelvalue, wait_milliseconds=.int)
	public	**.channelrequest_impl**	**start_structured**	(envelope=.channelvalue)
	public	**.completion**	**wait**	(wait_milliseconds=.int)
	public	**.void**	**cancel**	(request=.channelrequest, reason=.channelvalue)
	public	**.void**	**close**	(mode=.int)
	public	**.void**	**cancel_ticket**	(ticket=.int, reason=.channelvalue)
	public	**.completion_impl**	**wait_for**	(ticket=.int, wait_milliseconds=.int)
	public	**.channelvalue_impl**	**local_capability**	()
	public	**.boolean**	**open_state**	()
	public	**.int**	**pending**	()
	public	**.void**	**poll**	()
	public	**.void**	**cancel_pending**	(reason=.channelvalue)
	public	**.void**	**cancel_and_account**	()
*	public		**memory**	(direction=.int, capacity=.int)
*	public		**null_endpoint**	(direction=.int)
*	public		**from_reference**	(provider_reference=.channelvalue)
*	public		**from_encoded_reference**	(encoded=.binary)
	public	**.channelvalue**	**provider_reference**	()
	public	**.channelrequest**	**start_read**	(maximum_bytes=.int, wait_milliseconds=.int)
	public	**.channelrequest**	**start_write**	(bytes=.binary, wait_milliseconds=.int)
	public	**.void**	**half_close**	(direction=.int)
	public	**.void**	**cancel**	(reason=.channelvalue)

	public	.void	**close**	()
*	public		**binding**	(diagnostic_name=.string, signature=.string, binding=.binary, ?factory_arguments=.channelvalue[])
*	public		**sealed**	(diagnostic_name=.string, signature=.string, kind=.int, image_digest=.binary, callable_id=.int, adapter_callable_id=.int, signature_digest=.binary, factory_arguments=.channelvalue[])
	public	.string	**name**	()
	public	.string	**signature**	()
	public	.channelvalue_impl	**descriptor**	()
*	public		**empty**	()
	public	.taskarguments	**add_value**	(value=.channelvalue)
	public	.taskarguments	**add_channel_value**	(value=.channelvalue)
	public	.taskarguments	**add_boolean**	(value=.boolean)
	public	.taskarguments	**add_integer**	(value=.int)
	public	.taskarguments	**add_string**	(value=.string)
	public	.taskarguments	**add_binary**	(value=.binary)
	public	.channelvalue[]	**values**	()
*	public		**new**	(scope=reference .taskscope_-impl, channel=reference .channel_-impl, ticket=.int, identity=.int)
*	public		**rejected**	(scope=reference .taskscope_-impl, channel=reference .channel_-impl, identity=.int, cached=.completion_impl)
	public	.int	**identity**	()
	public	.boolean	**terminal**	()
	public	.void	**cancel**	(reason=.channelvalue)
	public	.completion	**wait**	(wait_milliseconds=.int)
	public	.completion	**completion**	()
*	public		**local**	(workers=.int, admission_capacity=.int)
*	public		**process**	(workers=.int, admission_capacity=.int)

	public	**.int**	**provider_type**	()
	public	**.int**	**capacity**	()
	public	**.int**	**admission_capacity**	()
	public	**.int**	**queued**	()
	public	**.int**	**running**	()
	public	**.void**	**close**	()
	public	**.channel_impl**	**pool_channel**	()
	public	**.boolean**	**open_state**	()
*	public		**default**	()
*	public		**failfast**	(pool=.taskpool, timeout_milliseconds=.int)
*	public		**collectall**	(pool=.taskpool, timeout_milliseconds=.int)
	public	**.task**	**submit**	(target=.tasktarget, request=.channelvalue)
	public	**.task**	**submit_arguments**	(target=.tasktarget, arguments=.taskarguments)
	public	**.task**	**ask**	(service=.serviceref, request=.channelvalue)
	public	**.completion**	**next**	(wait_milliseconds=.int)
	public	**.completion[]**	**join**	()
	public	**.void**	**cancel**	(reason=.channelvalue)
	public	**.void**	**close**	()
	public	**.void**	**finish**	()
	public	**.void**	**abort**	(reason=.channelvalue)
	public	**.int**	**result_integer**	(child=.task)
	public	**.string**	**result_string**	(child=.task)
	public	**.binary**	**result_binary**	(child=.task)
	public	**.channelvalue**	**result_value**	(child=.task)
	public	**.completion_impl**	**wait_task**	(identity=.int, wait_milliseconds=.int)
	public	**.void**	**mark_observed**	(identity=.int)
	public	**.int**	**pending**	()
	public	**.boolean**	**open_state**	()
*	public		**new**	(timeout_remaining=.int, cancellation_requested=.boolean, trace_identity=.string)
	public	**.int**	**timeout_remaining**	()
	public	**.boolean**	**cancellation_requested**	()
	public	**.string**	**trace_identity**	()
	public	**.byteendpoint**	**endpoint**	(provider_reference=.channelvalue)

.ArrayList

data

- .object
  - .ArrayList

*	public		**ArrayList**	()

public	.int	**init**	()
public	.int	**add**	(item=.object)
public	.int	**append**	(item=.object)
public	.int	**prepend**	(item=.object)
public	.object	**get**	(index=.int)
public	.int	**set**	(index=.int, item=.object)
public	.int	**insert**	(index=.int, item=.object)
public	.object	**remove**	(index=.int)
public	.int	**size**	()
public	.int	**isEmpty**	()
public	.ArrayListIterator	**iterator**	()
public	.ArrayListIterator	**snapshotIterator**	()
public	.void	**clear**	()

.ArrayListIterator — data

- .object
  - ∟ .ArrayListIterator

*	public		**ArrayListIterator**	(values=reference .object[], count=reference .int, snapshot_flag=.int)
	public	.int	**hasNext**	()
	public	.object	**next**	()
	public	.int	**index**	()
	public	.void	**reset**	()
	public	.int	**isLive**	()

.Bag — data

- *.Bag*

*	public		**Bag**	()
	public	.void	**add**	(value=.string)
	public	.int	**size**	()
	public	.string	**at**	(position=.int)
	public	.BagIterator	**– liveIterator**	()
	public	.BagIterator	**– snapshotIterator**	()

.ArrayBag — data

- .object
  - ∟ .ArrayBag

*	public		**ArrayBag**	()
	public	.void	**add**	(value=.string)
	public	.int	**size**	()
	public	.string	**at**	(position=.int)

public	**.ArrayBagIterator**	**liveIterator**	()
public	**.ArrayBagIterator**	**snapshotIterator**	()
public	**.string**	**toString**	()

.ArrayBagIterator — data

- .object
  - ∟ .ArrayBagIterator

*	public		**ArrayBagIterator**	(source=reference .ArrayBag, make_snapshot=.int)
	public	**.int**	**hasNext**	()
	public	**.string**	**next**	()

.Comparator — data

- *.Comparator*

public	**.int**	**compare**	(left=.object, right=.object)

.Iterable — data

- *.Iterable*

public	**.Iterator**	**iterator**	()

.Iterator — data

- *.Iterator*

public	**.int**	**hasNext**	()

.ObjectLinkedList — data

- .object
  - ∟ .ObjectLinkedList

*	public		**ObjectLinkedList**	()
	public	**.int**	**append**	(value=.object)
	public	**.int**	**prepend**	(value=.object)
	public	**.int**	**insertAfterCurrent**	(value=.object)
	public	**.int**	**insertBeforeCurrent**	(value=.object)
	public	**.int**	**remove**	()
	public	**.object**	**current**	()
	public	**.object**	**next**	()
	public	**.object**	**previous**	()
	public	**.int**	**setFirst**	()
	public	**.int**	**setLast**	()
	public	**.int**	**setPosition**	(pos=.string)
	public	**.int**	**clear**	()
	public	**.int**	**size**	()
	public	**.int**	**isEmpty**	()

public	**.object[]**	**toArray**	()
public	**.int**	**fromArray**	(list=.object[])

.TreeMap data

○ .object
  ∟ .TreeMap

*	public		**TreeMap**	(comp=.Comparator)
	public	**.int**	**put**	(key=.object, value=.object)
	public	**.object**	**get**	(key=.object)
	public	**.int**	**size**	()
	public	**.int**	**remove**	(key=.object)
	public	**.int**	**containsKey**	(key=.object)
	public	**.object**	**firstKey**	()
	public	**.object**	**lastKey**	()
	public	**.object[]**	**keyArray**	()
	public	**.object[]**	**valueArray**	()
	public	**.string**	**toString**	()
	public	**.string**	**String**	()
	public	**.TreeMapIterator**	**iterator**	()
	public	**.int**	**free**	()

.TreeMapIterator data

○ .object
  ∟ .TreeMapIterator

*	public		**TreeMapIterator**	(keys=.object[], values=.object[])
	public	**.int**	**hasNext**	()
	public	**.object**	**next**	()
	public	**.object**	**nextKey**	()
	public	**.object**	**value**	()
	public	**.object**	**nextValue**	()

.Printable data

○ *.Printable*

public	**.string**	**toString**	()

.ObjectStack data

○ .object
  ∟ .ObjectStack

*	public		**ObjectStack**	()
	public	**.int**	**push**	(item=.object)
	public	**.int**	**size**	()

	public	.int	**isEmpty**	()
	public	.int	**empty**	()
	public	.object	**pop**	()
	public	.object	**peek**	()

.os — system_os

- .object
  - ∟ .os

*	public		**os**	()
	public	.string	**uptime**	()
	public	.string	**user**	()
	public	.string	**loadpath**	()
	public	.string	**osname**	()
	public	.int	**appendBinaryFile**	(source=.string, target=.string)
	public	.string	**getHost**	()
	public	.int	**wait**	(time=.int)
	public	.int	**listDir**	(file=.string, entries=.string[])
	public	.string	**getEnv**	(env_name=.string)

.matchResult — data_qfind

- *.matchResult*

*	public		**matchResult**	(line=.string, delim=.string)
	public	.int	**position**	()

.fastFind — data_qfind

- .object
  - ∟ .fastFind

*	public		**matchResult**	(line=.string, delim=.string)
	public	.int	**position**	()
*	public		**fastFind**	(line=.string, delim=.string)
	public	.int	**found**	()
	public	.int	**position**	()
	public	.string	**before**	()
	public	.string	**match**	()
	public	.string	**after**	()

.qFind — data_qfind

- .object
  - ∟ .qFind

*	public		**matchResult**	(line=.string, delim=.string)
	public	.int	**position**	()
*	public		**fastFind**	(line=.string, delim=.string)
	public	.int	**found**	()
	public	.int	**position**	()
	public	.string	**before**	()
	public	.string	**match**	()
	public	.string	**after**	()
*	public		**qFind**	(line=.string, delim=.string)
	public	.int	**found**	()
	public	.int	**position**	()
	public	.string	**before**	()
	public	.string	**match**	()
	public	.string	**after**	()

.rexx

rexx

∘ .object
  ∟ .rexx

*	public		**rexx**	(s=.string)
	public	.boolean	**abbrev**	(astr=.string, len=0)
	public	.rexx	**abs**	()
	public	.rexx	**b2x**	()
	public	.int	**b2d**	()
	public	.rexx	**c2x**	()
	public	.int	**c2d**	()
	public	.rexx	**center**	(expose centlen=.int, pad=" ")
	public	.rexx	**centre**	(expose centlen=.int, pad=" ")
	public	.rexx	**changestr**	(needle=.string, nneedle=.string)
	public	.int	**compare**	(astr=.string, pad=" ")
	public	.rexx	**copies**	(count=.int)
	public	.int	**countstr**	(needle=.string)
	public	.rexx	**d2b**	()
	public	.rexx	**d2c**	()
	public	.rexx	**d2x**	()
	public	.rexx	**delstr**	(position=.int, dellen=0)
	public	.rexx	**delword**	(wnum=.int, wcount=0)

public	.rexx	**format**	(before=0, after=0, expp=0, expt=-1)
public	.rexx	**insert**	(string=.string, position=0, len=0, pad=" ")
public	.int	**length**	()
public	.int	**linesize**	()
public	.rexx	**lower**	()
public	.rexx	**overlay**	(string=.string, position=1, len=0, pad=" ")
public	.int	**pos**	(needle=.string, start=1)
public	.int	**lastpos**	(needle=.string, search_end=0)
public	.rexx	**left**	(number=.int, pad=" ")
public	.rexx	**right**	(number=.int, pad=" ")
public	.rexx	**reverse**	()
public	.rexx	**sign**	()
public	.rexx	**space**	(spacenr=1, pad=" ")
public	.rexx	**strip**	(option="B", schar=" ")
public	.rexx	**substr**	(start=.int, len=0, pad=" ")
public	.string	**symbol**	()
public	.rexx	**translate**	(tochar="", fromchar="", pad=" ")
public	.rexx	**trunc**	(fraction=0)
public	.rexx	**upper**	()
public	.rexx	**value**	()
public	.int	**verify**	(intab=.string, match_mode='N', spos=1)
public	.rexx	**word**	(wordnum=.int)
public	.int	**words**	()
public	.int	**wordindex**	(wordnum=.int)
public	.rexx	**subword**	(wordnum=.int, wordcount=0)

public	.int	**wordlength**	(wordnum=.int)
public	.int	**wordpos**	(string=.string, start=1)
public	.rexx	**x2b**	()
public	.rexx	**x2c**	()
public	.int	**x2d**	(signed_length=0)
public	.rexx	**reradix**	(fromRadix=.int, toRadix=.int)
public	.rexx	**sequence**	(tos=.string)
public	.rexx	**xrange**	(tos=.string)
public	.string	**datatype**	(type="")
public	.string	**stringValue**	()
public	.string	**toString**	()

.RexxComparator — data

- .object
  - ∟ .RexxComparator

*	public		**RexxComparator**	()
	public	.int	**compare**	(left=.object, right=.object)

.scanlex_token — scanlex

- .object
  - ∟ .scanlex_token

*	public		**scanlex_token**	(type=.string, text=.string, pos=.int)
	public	.string	**get_type**	()
	public	.string	**get_text**	()
	public	.int	**get_pos**	()

.StringArrayList — data

- .object
  - ∟ .StringArrayList

*	public		**StringArrayList**	()
	public	.int	**init**	()
	public	.int	**add**	(item=.string)
	public	.int	**append**	(item=.string)
	public	.int	**prepend**	(item=.string)
	public	.string	**get**	(index=.int)
	public	.int	**set**	(index=.int, item=.string)
	public	.int	**insert**	(index=.int, item=.string)

public	.string	**remove**	(index=.int)
public	.int	**indexOf**	(item=.string, from=1)
public	.int	**contains**	(item=.string)
public	.int	**size**	()
public	.int	**isEmpty**	()
public	.StringArrayListIterator	**iterator**	()
public	.StringArrayListIterator	**snapshotIterator**	()
public	.void	**clear**	()

.StringArrayListIterator data

- .object
  - ∟ .StringArrayListIterator

*	public		**StringArrayListIterator**	(values=reference .string[], count=reference .int, snapshot_flag=.int)
	public	.int	**hasNext**	()
	public	.string	**next**	()
	public	.int	**index**	()
	public	.void	**reset**	()
	public	.int	**isLive**	()

.StringHashMap data

- .object
  - ∟ .StringHashMap

*	public		**StringHashMap**	(expected=1024)
	public	.int	**_find**	(key=.string)
	public	.int	**put**	(key=.string, value=.string)
	public	.string	**get**	(key=.string)
	public	.int	**containsKey**	(key=.string)
	public	.int	**remove**	(key=.string)
	public	.int	**size**	()
	public	.string[]	**keyArray**	()
	public	.string[]	**valueArray**	()
	public	.StringTreeSet	**keySet**	()
	public	.StringTreeSet	**valueSet**	()
	public	.StringHashMapIterator	**iterator**	()
	public	.string	**toString**	()
	public	.int	**free**	()

.StringHashMapIterator data

- .object
  - ∟ .StringHashMapIterator

*	public		**StringHashMapIterator**	(keys=.string[], values=.string[])
	public	**.int**	**hasNext**	()
	public	**.string**	**next**	()
	public	**.string**	**nextKey**	()
	public	**.string**	**nextValue**	()
	public	**.int**	**close**	()

.StringHashSet data

- .object
  - ∟ .StringHashSet

*	public		**StringHashSet**	(expected=1024)
	public	**.int**	**_find**	(key=.string)
	public	**.int**	**add**	(key=.string)
	public	**.int**	**fromArray**	(items=.string[])
	public	**.int**	**contains**	(key=.string)
	public	**.int**	**remove**	(key=.string)
	public	**.int**	**size**	()
	public	**.string[]**	**toArray**	()
	public	**.StringHashSetIterator**	**iterator**	()
	public	**.string**	**toString**	()
	public	**.int**	**free**	()

.StringHashSetIterator data

- .object
  - ∟ .StringHashSetIterator

*	public		**StringHashSetIterator**	(items=.string[])
	public	**.int**	**hasNext**	()
	public	**.string**	**next**	()
	public	**.int**	**close**	()

.StringIterable data

- *.StringIterable*

public	**.StringIterator**	**iterator**	()

.StringIterator data

- *.StringIterator*

public	**.int**	**hasNext**	()

.StringLinkedList data

- .object
  - ∟ .StringLinkedList

*	public		**StringLinkedList**	()
	public	**.int**	**append**	(value=.string)
	public	**.int**	**prepend**	(value=.string)
	public	**.int**	**insertAfterCurrent**	(value=.string)
	public	**.int**	**insertBeforeCurrent**	(value=.string)
	public	**.int**	**remove**	()
	public	**.string**	**current**	()
	public	**.string**	**next**	()
	public	**.string**	**previous**	()
	public	**.int**	**currentAddress**	()
	public	**.int**	**setFirst**	()
	public	**.int**	**setLast**	()
	public	**.int**	**setPosition**	(pos=.string)
	public	**.int**	**setAddress**	(addr=.int)
	public	**.int**	**clear**	()
	public	**.int**	**size**	()
	public	**.int**	**isEmpty**	()
	public	**.string[]**	**toArray**	()
	public	**.int**	**fromArray**	(list=.string[])
	public	**.int**	**debug**	()
	public	**.string**	**toString**	()

.StringObjectHashMap data

- .object
  - ∟ .StringObjectHashMap

*	public		**StringObjectHashMap**	(expected=1024)
	public	**.int**	**_find**	(key=.string)
	public	**.int**	**put**	(key=.string, value=.object)
	public	**.object**	**get**	(key=.string)
	public	**.int**	**containsKey**	(key=.string)
	public	**.int**	**remove**	(key=.string)
	public	**.int**	**size**	()
	public	**.string[]**	**keyArray**	()
	public	**.object[]**	**valueArray**	()
	public	**.StringTreeSet**	**keySet**	()
	public	**.StringObjectHashMapIterator**	**iterator**	()
	public	**.int**	**_indexToken**	()
	public	**.int**	**free**	()

.StringObjectHashMapIterator data

- .object
  - ∟ .StringObjectHashMapIterator

*	public		**StringObjectHashMapIterator**	(keys=.string[], values=.object[])

	public	.int	hasNext	()
	public	.string	next	()
	public	.string	nextKey	()
	public	.object	value	()
	public	.object	nextValue	()
	public	.int	close	()

.StringObjectTreeMap data

- .object
  - ∟ .StringObjectTreeMap

*	public		**StringObjectTreeMap**	()
	public	**.int**	**put**	(key=.string, value=.object)
	public	**.object**	**get**	(key=.string)
	public	**.int**	**size**	()
	public	**.int**	**remove**	(key=.string)
	public	**.int**	**containsKey**	(key=.string)
	public	**.string**	**firstKey**	()
	public	**.string**	**lastKey**	()
	public	**.string[]**	**keyArray**	()
	public	**.object[]**	**valueArray**	()
	public	**.StringTreeSet**	**keySet**	()
	public	**.StringObjectTreeMapIterator**	**iterator**	()
	public	**.string**	**toString**	()
	public	**.string**	**String**	()
	public	**.int**	**free**	()

.StringObjectTreeMapIterator data

- .object
  - ∟ .StringObjectTreeMapIterator

*	public		**StringObjectTreeMapIterator**	(keys=.string[], values=.object[])
	public	**.int**	**hasNext**	()
	public	**.string**	**next**	()
	public	**.string**	**nextKey**	()
	public	**.object**	**value**	()
	public	**.object**	**nextValue**	()
	public	**.int**	**close**	()

.StringOldTreeMap data

- .object
  - ∟ .StringOldTreeMap

*	public		**StringOldTreeMap**	()
	public	**.int**	**_find**	(key=.string)

public	.string[]	__sortedKeys	()
public	.int	put	(key=.string, value=.string)
public	.string	get	(key=.string)
public	.int	size	()
public	.int	remove	(key=.string)
public	.int	containsKey	(key=.string)
public	.string	firstKey	()
public	.string	lastKey	()
public	.string[]	keyArray	()
public	.string[]	valueArray	()
public	.StringTreeSet	keySet	()
public	.StringTreeSet	valueSet	()
public	.StringTreeMapIterator	iterator	()
public	.string	toString	()
public	.int	free	()

.StringStack

data

- .object
  - .StringStack

*	public		StringStack	()
	public	.int	push	(item=.string)
	public	.int	size	()
	public	.int	isEmpty	()
	public	.int	empty	()
	public	.string	pop	()
	public	.string	peek	()

.StringTreeMap

data

- .object
  - .StringTreeMap

*	public		StringTreeMap	()
	public	.int	put	(key=.string, value=.string)
	public	.string	get	(key=.string)
	public	.int	size	()
	public	.int	remove	(key=.string)
	public	.int	containsKey	(key=.string)
	public	.string	firstKey	()
	public	.string	lastKey	()
	public	.string[]	keyArray	()
	public	.string[]	valueArray	()
	public	.StringTreeSet	keySet	()
	public	.StringTreeSet	valueSet	()
	public	.StringTreeMapIterator	iterator	()

public	.string	**toString**	()
public	.string	**String**	()
public	.int	**free**	()

.StringTreeMapIterator data

- .object
  - .StringTreeMapIterator

*	public		**StringTreeMapIterator**	(keys=.string[])
	public	.int	**hasNext**	()
	public	.string	**next**	()
	public	.int	**close**	()

.StringTreeSet data

- .object
  - .StringTreeSet

*	public		**StringTreeSet**	()
	public	.int	**add**	(key=.string)
	public	.int	**fromArray**	(items=.string[])
	public	.int	**size**	()
	public	.int	**remove**	(key=.string)
	public	.int	**contains**	(key=.string)
	public	.string	**first**	()
	public	.string	**last**	()
	public	.string[]	**toArray**	()
	public	.StringTreeSetIterator	**iterator**	()
	public	.string	**toString**	()
	public	.int	**free**	()

.StringTreeSetIterator data

- .object
  - .StringTreeSetIterator

*	public		**StringTreeSetIterator**	(items=.string[])
	public	.int	**hasNext**	()
	public	.string	**next**	()
	public	.int	**close**	()

.id id_id

- .object
  - .id

*	public		**id**	()
	public	.string	**uuid**	()
	public	.string	**uuidV7**	()
	public	.string	**ulid**	()

public	**.string**	**nanoId**	()
public	**.string**	**snowflake**	()
public	**.string**	**base58**	()

.KeyDB — data_KeyDB

- .object
  - ∟ .KeyDB

*	public		**KeyDB**	()
	public	**.int**	**open**	(filename=.string, mode=.string)
	public	**.int**	**close**	()
	public	**.int**	**isOpen**	()
	public	**.int**	**begin**	()
	public	**.int**	**commit**	()
	public	**.int**	**rollback**	()
	public	**.int**	**put**	(key=.string, value=.string)
	public	**.string**	**get**	(key=.string)
	public	**.int**	**remove**	(key=.string)
	public	**.int**	**size**	()
	public	**.string**	**stats**	()
	public	**.int**	**backup**	(path=.string)
	public	**.int**	**validate**	()
	public	**.int**	**compact**	()
	public	**.int**	**containsKey**	(key=.string)
	public	**.int**	**putCommit**	(key=.string, value=.string)
	public	**.int**	**removeCommit**	(key=.string)
	public	**.int**	**getHandle**	()
	public	**.int**	**dispose**	()

.DateTime — time

- .object
  - ∟ .DateTime

*	public		**DateTime**	(base_date=date("BASE") as .int, , day_microseconds=time("US") as .int)
	public	**.int**	**baseDate**	()
	public	**.int**	**dayMicroseconds**	()
	public	**.int**	**daySeconds**	()
	public	**.int**	**year**	()
	public	**.int**	**month**	()
	public	**.int**	**day**	()

public · · · · · · · · · · · ·	.int	**hour**	()
public · · · · · · · · · · · ·	.int	**minute**	()
public · · · · · · · · · · · ·	.int	**second**	()
public · · · · · · · · · · · ·	.int	**microsecond**	()
public · · · · · · · · · · · ·	.int	**yearDay**	()
public · · · · · · · · · · · ·	.string	**dayName**	()
public · · · · · · · · · · · ·	.string	**monthName**	()
public · · · · · · · · · · · ·	.string	**standardDate**	()
public · · · · · · · · · · · ·	.string	**internationalDate**	()
public · · · · · · · · · · · ·	.int	**isLeapYear**	()
public · · · · · · · · · · · ·	.int	**daysInYear**	()
public · · · · · · · · · · · ·	.int	**daysInMonth**	()
public · · · · · · · · · · · ·	.int	**addMicroseconds**	(microseconds=.int)
public · · · · · · · · · · · ·	.int	**addSeconds**	(seconds=.int)
public · · · · · · · · · · · ·	.int	**addMinutes**	(minutes=.int)
public · · · · · · · · · · · ·	.int	**addHours**	(hours=.int)
public · · · · · · · · · · · ·	.int	**addDays**	(days=.int)
public · · · · · · · · · · · ·	.int	**addWeeks**	(weeks=.int)
public · · · · · · · · · · · ·	.int	**compareTo**	(other=.DateTime)
public · · · · · · · · · · · ·	.int	**isBefore**	(other=.DateTime)
public · · · · · · · · · · · ·	.int	**isAfter**	(other=.DateTime)
public · · · · · · · · · · · ·	.int	**addYears**	(years=.int)
public · · · · · · · · · · · ·	.int	**weekDay**	()
public · · · · · · · · · · · ·	.int	**weekNumberYear**	()
public · · · · · · · · · · · ·	.int	**weekNumber**	()
public · · · · · · · · · · · ·	.string	**weekNumberDate**	()
public · · · · · · · · · · · ·	.int	**weeksInYear**	()
public · · · · · · · · · · · ·	.int	**equals**	(other=.DateTime)
public · · · · · · · · · · · ·	.string	**toString**	()

.ISO8601 time

- .object
  - ∟ .ISO8601

*	public · · · · · · · · · · · ·		**ISO8601**	()
	public · · · · · · · · · · · ·	.string	**format**	(value=.DateTime)
	public · · · · · · · · · · · ·	.string	**formatDate**	(value=.DateTime)
	public · · · · · · · · · · · ·	.string	**formatSeconds**	(value=.DateTime)
	public · · · · · · · · · · · ·	.DateTime	**parse**	(text=.string)
	public · · · · · · · · · · · ·	.int	**isValid**	(text=.string)
	public · · · · · · · · · · · ·	.OffsetDateTime	**– parseOffset**	()
	public · · · · · · · · · · · ·	.OffsetDateTime	**parseOffset**	(text=.string)
	public · · · · · · · · · · · ·	.string	**formatOffset**	(value=.OffsetDateTime)

.OffsetDateTime time

- .object

⌞ .OffsetDateTime

*	public · · · · · · · · · · · ·		**OffsetDateTime**	(datetime=.DateTime(), offset_minutes=0)
	public · · · · · · · · · · · ·	**.DateTime**	**dateTime**	()
	public · · · · · · · · · · · ·	**.int**	**offsetMinutes**	()
	public · · · · · · · · · · · ·	**.OffsetDateTime**	**– toUTC**	()
	public · · · · · · · · · · · ·	**.OffsetDateTime**	**toUTC**	()

PART III

Reference

7

Level B Built-in functions

7.1 ADDRESS environment protocol

`_address.crexx` is the Level B runtime protocol behind the `ADDRESS` statement, ADDRESS function calls, host callbacks, redirects, and variable writeback. It is not the Classic Level C `ADDRESS()` query BIF; that separate contract is documented in `lib/rxfnsc/address.md`.

7.1.1 Main surfaces

The module provides four related layers:

1. `addressdriverregistry` normalizes driver names, aliases, and prefixes and resolves a requested environment to its registered driver.
2. `addressstem/standardaddressstem` and `addresssandbox/standardaddresssandbox` provide normalized key/value stores. Their `next(cursor)` cursor is an `.int`; zero starts iteration.
3. `addressbinding`, `addressrequest`, `addressresponse`, `addressfunctionrequest`, and `addressfunctionresponse` carry commands, functions, redirects, sandbox data, exposed values/stems, return codes, and diagnostics.
4. `_address*`, `_new_address_environment`, `_ensure_address_environment`, `_register_address_environment`, and `_set_address_environment` construct, dispatch, register, and select environment implementations.

Environment names and sandbox/stem keys are stripped and uppercased for lookup. Commands, values, diagnostics, redirect resources, and provider data remain strings. Counts, indexes, return codes, binding slots, and iteration cursors use `.int` at the Level B boundary.

The request/response, binding, standard sandbox, and standard stem class attribute order is part of the native VM bridge ABI in `interpreter/rxvml.c`. Do not reorder or insert attributes without changing and testing that bridge.

7.1.2 Errors

Invalid programmer arguments are reported with the `INVALID_ARGUMENTS` signal. In particular, driver and environment names must not be blank, and a stem binding count passed to the native spawn bridge must not be negative. Runtime provider failures remain `addressresponse` or `addressfunctionresponse` values with a return code, condition name, and optional diagnostics.

An unknown command environment produces a failure response; it is not an argument-type error.

7.1.3 Examples

The following operations do not launch an external command:

```
registry = .addressdriverregistry()
call registry.add("openai", "gpt", "gpt_")
say registry.lookup("gpt_4_1")  /* OPENAI */

sandbox = .standardaddresssandbox()
call sandbox.set("Mode", "safe")
say sandbox.get("MODE")         /* safe */
say sandbox.next(0)             /* MODE */
```

To handle a bad name:

```
do
  call _set_address_environment("")
on signal invalid_arguments as problem
  say problem.name()             /* INVALID_ARGUMENTS */
end
```

7.1.4 Performance notes

Registry entries and stored keys are normalized on insertion. Lookups normalize the search key once and compare it with the stored normalized keys; they do not repeatedly strip and uppercase every stored entry. Updating an existing sandbox or stem entry reuses the normalized key rather than normalizing it again. The current registries are compact arrays, so lookup is linear in the number of registered entries and performs no per-entry normalization or object allocation.

7.1.5 Coverage

`lib/rxfnsb/tests_functional/ts_address_protocol.crexx` covers the pure Level B protocol in optimized and unoptimized modes: registry aliases/prefixes, typed cursors, normalization, drop/iteration, request/response objects, sandbox/stem/scalar writeback, function requests, environment selection, unknown-environment responses, and signal errors. The existing `ts_address.crexx`, `ts_address_crexx.crexx`, compiler ADDRESS cases, and native host callback tests cover execution integration.

7.2 _datei

`_datei(idate=.string, format=.string [,isep=""]) -> .int` is the private input half of the extended Level B DATE implementation. It converts one format documented in `date.md` to a validated Julian day number.

The implementation performs direct Unicode-codepoint scans for integer, fixed-width, separated, and named-month forms. It does not call general WORD, SUBSTR, POS, RIGHT, or abbreviation BIFs on the parsing path. Two-digit years map to 2000 through 2099; X formats require four digits; QUALIFIED input checks that the supplied weekday matches the date. Negative EPOCH values use floor division so `-1` identifies 1969-12-31.

Malformed text, unknown/output-only formats, invalid dates, out-of-range origins, and a separator longer than one codepoint signal `INVALID_ARGUMENTS`. The public `date()` wrapper also validates separator length before dispatch.

7.3 _dateo

`_dateo(jdn=.int, format=.string [,osep=""]) -> .string` is the private output half of the extended Level B DATE implementation. Its formats and separator defaults are documented in `date.md`.

The helper validates the JDN once, derives all Gregorian fields once, and then formats with direct digit/codepoint append operations. It does not route numeric fields through general RIGHT, WORD, or SUBSTR calls. Every branch returns `.string`, including the signed UNIX/EPOCH and numeric BASE/JDN forms.

An invalid JDN, output format, or separator longer than one codepoint signals `INVALID_ARGUMENTS`; there is no fallback format.

7.4 _ftrunc

`_ftrunc` is a private `_rxsysb` helper used by Level B `format`:

```
_ftrunc(number = .float) = .string
```

It converts `number` with the VM's active numeric context and returns the text after the decimal point. If the formatted representation has no decimal point, it returns the empty string.

```
_ftrunc(12.75)  /* "75" */
_ftrunc(-0.125) /* "125" */
_ftrunc(42.0)   /* "" */
```

Typed-call conversion failures are reported through the VM's `CONVERSION_ERROR` signal. The caller's value is not modified.

After the numeric conversion, the implementation performs one native decimal-point search and at most one direct substring. It has no interpreted per-character loop, helper call, or append loop.

This is not a public Classic BIF and has no Level C contract. Its output is an implementation component of `format`, not an independent decimal-arithmetic API.

7.5 _itrunc

`_itrunc` is a private `_rxsysb` helper used by Level B `format`:

```
_itrunc(number = .float) = .string
```

It converts `number` with the VM's active numeric context and returns the text before the decimal point, including a negative sign. If the formatted representation has no decimal point, it returns that representation directly.

```
_itrunc(12.75)  /* "12" */
_itrunc(-0.125) /* "-0" */
_itrunc(42.0)   /* "42" */
```

Typed-call conversion failures are reported through the VM's `CONVERSION_ERROR` signal. The caller's value is not modified.

After numeric conversion, the implementation performs one native decimal-point search and at most one direct substring. It has no interpreted per-character loop, helper call, conversion-per-character, or append loop.

This is not a public Classic BIF and has no Level C contract. Its output is an implementation component of `format`, not an independent decimal-arithmetic API.

7.6 DATE calendar core

`_rxsysb` exposes the typed proleptic-Gregorian primitives used by the Level B DATE cluster and the standalone Classic DATE adapter:

```
_jdn(day=.int, month=.int, year=.int) = .int
_datefromjdn(jdn=.int, expose year=.int, expose month=.int,
             expose day=.int, expose day_of_year=.int,
             expose weekday=.int, expose base_day=.int)
_datevalid(day=.int, month=.int, year=.int) = .boolean
_leapyear(year=.int) = .boolean
_daysinmonth(month=.int, year=.int) = .int
```

The supported calendar is Gregorian year 1 through 9999. JDN 1721426 is 0001-01-01 and JDN 5373484 is 9999-12-31. Weekdays use Monday=1 through Sunday=7; `base_day` is zero on 0001-01-01.

`_jdn` and `_datefromjdn` signal `INVALID_ARGUMENTS` outside this domain. `_datefromjdn` computes into locals and does not change its exposed outputs when validation fails. `_daysinmonth` returns zero for an invalid year/month.

The conversion is constant-time integer arithmetic. The focused `ts_date_core` harness covers known JDN values, Gregorian century/leap rules, round trips, both boundaries, signals, and exposed-output preservation.

7.7 System Primitives

`_rxsystem.crexx` supplies three low-level Level B runtime helpers. They are support functions used by the Level B file library, not Classic Level C BIFs.

7.8 `_exit`

```
call _exit(return_code)
```

`return_code` is an `.int` and defaults to zero when omitted. `_exit` terminates the VM process with that exact code and does not return.

7.8.1 _open

```
fileid = _open(name, mode)
```

`name` and `mode` are strings; the result is an `.int` VM file handle. Mode is one of these lowercase bootstrap tokens:

Mode	Cached stream behavior
`r`	read
`w`	append-capable output
`a`	append-capable output

The `w` and `a` spellings share one cached C append mode. This is intentional: Level B `lineout` and `charout` keep the handle open between calls and must not truncate the stream on each call. Code that needs explicit truncation uses a direct write-mode open, as `eraseFile` does.

Repeated opens of the same name and canonical mode return the cached handle. Changing between read and output closes the old handle and reopens the file. The first free cache slot is reused. A failed OS open returns zero and is not cached.

Blank names and modes other than `r`, `w`, or `a` raise `INVALID_ARGUMENTS`. A failure while closing for a mode change raises `NOTREADY`.

7.8.2 `_close`

```
call _close(name [, nomsg])
```

`name` is a string and `nomsg` is an optional `.int` flag restricted to 0 or 1 and defaulting to 0. This preserves the established optional-call surface while avoiding string coercion at the Level B boundary. The cached handle and all of its cache metadata are cleared. A blank name raises `INVALID_ARGUMENTS`; a close failure or a name that is not open raises `NOTREADY`. Passing 1 for `nomsg` makes a missing/already-closed name a no-op and suppresses a native close failure. This form is used only by cleanup paths such as `eraseFile`.

7.8.3 Example

```
fileid = _open("events.log", "w")
assembler fwrite fileid,"started"
call _close("events.log")
```

7.8.4 Coverage and performance

`lib/rxfnsb/tests_functional/ts_rxsystem.crexx` exercises handle reuse, mode changes, data preservation, failed-open cache behavior, typed suppression, and the documented signals in opt and noopt modes. `ts_rxsystem_exit.crexx` checks the exact integer process status in both modes.

The file cache remains a compact linear array because normal Level B programs hold few simultaneous files. Searches allocate nothing, compare each stored name once, reuse the first empty slot, canonicalize output modes before lookup, and never retain a failed handle.

7.9 abbrev

`abbrev` tests a case-sensitive leading abbreviation:

```
abbrev(string = .string, candidate = .string, minimum = 0) = .int
```

It returns `1` only when all of these are true:

- `candidate` is no longer than `string`;
- `candidate` contains at least `minimum` characters; and

- every candidate character equals the corresponding leading character of `string`.

The default minimum is zero. For an omitted minimum this produces the same prefix result as the Classic default of the candidate length. An empty candidate therefore matches unless a positive minimum is supplied.

`minimum` is an integer Level B argument. A negative value raises the `INVALID_-ARGUMENTS` signal rather than being silently accepted.

```
abbrev("PRINT", "PRI")     /* 1 */
abbrev("PRINT", "Pri")     /* 0 */
abbrev("PRINT", "PRI", 4)  /* 0 */
abbrev("PRINT", "")        /* 1 */
```

The implementation compares codepoints directly and checks both lengths before reading either string. `lib/rxfnsb/tests_functional/ts_abbrev.crexx` covers the examples, short-source safety, Unicode, exact minimum boundaries, qualified calls, and the negative-length signal.

7.10 abs

The native Level B `abs` function returns a non-negative decimal value:

```
abs(number = .decimal) = .decimal
```

```
abs(12.3)           /* 12.3 */
abs(-12.345)        /* 12.345 */
abs("-123.45E+16") /* 1.2345E+18 after typed conversion */
```

The Level B call boundary performs the ordinary `.decimal` conversion. The function itself needs only one comparison and, for a negative value, one decimal subtraction. It allocates no string, calls no helper, and does not modify the caller's value.

Invalid dynamic conversion raises the catchable `CONVERSION_ERROR` signal.

This is the strongly typed foundation API. It does not perform Classic Rexx numeric-text cleanup itself. The separate Level C `ABS` BIF accepts and normalizes Classic numeric text through its `rNUM` contract.

7.11 arrayappend

```
arrayappend(array = .string[] expose,
            value = .string,
            count = .int optional) = .int
```

`arrayappend` adds `value` to the end of `array` `count` times and returns the new high-water mark. `count` defaults to one. The array uses `array[0]` as its automatically maintained high-water mark.

A zero count is a no-op. A negative count raises `INVALID_ARGUMENTS`.

The exposed array is mutated in place. The implementation uses one VM `insattrs1` bulk growth operation followed by exactly one value assignment per new slot; it does not move the existing elements in Rexx code. The focused harness is `lib/rxfnsb/tests_functional/ts_arrayappend.crexx`.

7.12 arraycontains

```
arraycontains(array = .string[], value = .string [, case = .int]) = .
    int
```

`arraycontains` returns `1` when a complete array element equals `value`, and `0` when no element matches. It searches from index one through `array[0]` and returns on the first match. Empty strings are ordinary searchable values.

`case` defaults to `1` for case-sensitive string equality. Supply `0` for uppercase-folded matching. Any other value signals `INVALID_ARGUMENTS`.

The array is aliased read-only rather than value-copied and is never mutated. For insensitive search the implementation folds `value` once, reuses one candidate buffer, and calls the VM uppercase operation directly. The focused harness is `lib/rxfnsb/tests_functional/ts_arraycontains.crexx`.

7.13 arraycopy

```
arraycopy(array = .string[] expose [, from = .int [, count = .int]]) =
    .string[]
```

`arraycopy` returns a new array containing up to `count` strings starting at `from`. `from` defaults to one. A negative start counts back from the end (`-1` is the last element); zero or a start outside the source returns an empty array.

count defaults to zero, meaning copy through the end. A larger count clamps to the available tail. A negative count signals INVALID_ARGUMENTS.

The source is aliased read-only rather than value-copied, and changing the returned array does not change it. The implementation sizes the result once, then copies each selected string through direct linked attributes without per-element growth checks. The focused harness is lib/rxfnsb/tests_functional/ts_arraycopy.crexx.

7.14 arraydelete

```
arraydelete(array = .string[] expose,
            from = .int,
            count = .int) = .int
```

arraydelete removes up to count elements beginning at the positive one-based index from, shifts the remaining tail down, and returns the new high-water mark. The array uses array[0] as its automatically maintained high-water mark.

A zero count, empty array, or start beyond the high-water mark is a no-op. An overlong count clamps to the available tail without overflowing native index arithmetic. A position below one or negative count raises INVALID_ARGUMENTS.

The exposed array is mutated in place through one VM delattrs1 bulk pointer shift; the existing tail is not copied in a Rexx-level loop. The focused harness is lib/rxfnsb/tests_functional/ts_arraydelete_focused.crexx.

7.15 arraydrop

```
arraydrop(array = .string[] expose) = .int
```

arraydrop clears every logical element from a string array in place and returns zero, the new high-water mark. Clearing an empty array is an idempotent no-op, and the same array can be populated again afterward.

The implementation is one VM setattrs array,0 operation. It does not walk or copy string values in REXX code. The focused harness is lib/rxfnsb/tests_functional/ts_arraydrop.crexx.

7.16 arrayfind

```
arrayfind(needle = .string,
```

```
        array = .string[],
        from = .int optional,
        case_sensitive = .int optional) = .int
```

`arrayfind` returns the first one-based index at or after `from` whose element contains `needle`, or zero when no element matches. The string-array convention uses `array[0]` as the high-water mark. `from` defaults to one; a value below one clamps to one, while a value above the high-water mark returns zero.

`case_sensitive` defaults to `1`. Supply `0` for Unicode uppercase-folded matching. Any other flag raises `INVALID_ARGUMENTS`. An empty needle returns zero, matching the Level B `pos` contract.

The implementation reads but does not mutate the array. It uses direct VM substring search, folds the needle once for insensitive searches, and makes no other selector calls. The focused harness is `lib/rxfnsb/tests_functional/ts_arrayfind.crexx`.

7.17 arrayget

```
arrayget(array = .string[], index = .int [, default = .string]) = .
    string
```

`arrayget` returns `array[index]` when `index` is within the one-based range `1..array[0]`. It returns `default` when the array is empty or the index is outside that range. The omitted default is the empty string.

An out-of-range index is a successful fallback lookup, not an error. The input array is not exposed or modified. The implementation performs two bounds checks and reads the array element only on the in-range path.

The focused harness is `lib/rxfnsb/tests_functional/ts_arrayget.crexx`.

7.18 arrayhi

```
arrayhi(array = .string[] expose
        [, mode = .string [, newhi = .int]]) = .int
```

`arrayhi(array)` and case-insensitive `arrayhi(array, "GET")` return the current `array[0]` high-water mark without changing the array. `arrayhi(array, "SET", newhi)` shrinks the array when `newhi` is positive and smaller than its current mark. A same/larger or non-positive `newhi` is a documented no-op; `arrayhi` never grows an array.

An unknown mode, `SET` without `newhi`, or `GET` with `newhi` signals `INVALID_ARGUMENTS`. The omitted GET path returns immediately; explicit modes use one direct VM uppercase operation, and an actual shrink uses one `setattrs`.

The focused harness is `lib/rxfnsb/tests_functional/ts_arrayhi.crexx`.

7.19 arrayindexof

```
arrayindexof(array = .string[], value = .string
             [, from = .int [, case = .int]]) = .int
```

`arrayindexof` returns the first one-based index at or after `from` whose complete element equals `value`, or zero when no element matches. `from` defaults to one; a lower value clamps to one, while a value above `array[0]` returns zero. Empty strings are ordinary searchable values.

`case` defaults to `1` for case-sensitive string equality. Supply `0` for uppercase-folded matching. Any other value signals `INVALID_ARGUMENTS`.

The array is aliased read-only rather than value-copied and is never mutated. For insensitive search the implementation folds `value` once, reuses one candidate buffer, and calls the VM uppercase operation directly. The focused harness is `lib/rxfnsb/tests_functional/ts_arrayindexof.crexx`.

7.20 arrayinsert

```
arrayinsert(array = .string[] expose,
            from = .int,
            count = .int,
            default = .string optional) = .int
```

`arrayinsert` opens `count` slots at the positive one-based position `from`, shifts the existing tail upward, assigns `default` to every new slot, and returns the new high-water mark. `default` is the empty string when omitted. The array uses `array[0]` as its automatically maintained high-water mark.

A position beyond the end clamps to append. A zero count is a no-op. A position below one or negative count raises `INVALID_ARGUMENTS`.

The exposed array is mutated in place. The implementation uses one VM `insattrs1` bulk pointer shift followed by exactly one assignment per inserted slot; it does not copy the existing tail in Rexx code. The focused harness is `lib/rxfnsb/tests_-functional/ts_arrayinsert.crexx`.

7.21 arrayjoin

`arrayjoin` combines a one-based Level B string array without modifying it:

```
arrayjoin(array=.string[] [,separator=.string]) = .string
```

The separator is inserted between adjacent elements only and defaults to the empty string. Empty-string elements still participate in separator placement. An empty array returns empty; a one-element array returns that value unchanged.

```
items = .string[]
items[1] = "A"
items[2] = ""
items[3] = "C"
arrayjoin(items, "|") == "A||C"
```

The implementation reads the high-water mark once and appends each separator and element directly into one growing destination. It is O(total result text), creates no per-element concatenation result, and has no helper calls. Inputs are read-only. There is no Level C BIF or class method named ARRAYJOIN.

`lib/rxfnsb/tests_functional/ts_arrayjoin.crexx` covers empty/single arrays, empty/Unicode values and separators, omission, and non-mutation.

7.22 arraymove

```
arraymove(array = .string[] expose,
          from = .int, count = .int, to = .int) = .int
```

`arraymove` moves up to `count` strings beginning at the positive one-based `from` index so they begin at insertion position `to` in the original array. It mutates the array in place and returns its unchanged high-water mark.

The source count clamps to the available tail and a destination beyond the end clamps to append. Count zero, an empty array, a source above the high-water mark, and a destination within or immediately after the selected block are no-ops. A source or destination below one, or a negative count, signals `INVALID_-ARGUMENTS`.

The implementation inserts the destination gap first, making the source and target ranges non-overlapping. It then performs one direct string copy per moved value and one bulk deletion, without a temporary array. The focused harness is `lib/rxfnsb/tests_functional/ts_arraymove.crexx`.

7.23 arraypop

`arraypop` removes the final element of a one-based Level B string array:

```
arraypop(expose array=.string[] [,default=.string]) = .string
```

When the array is nonempty, its last value is returned and its high-water mark is reduced by one. Empty-string elements are values and are still removed. When the array is empty, it is unchanged and `default` is returned; the default value is the empty string.

```
items = .string[]
items[1] = "A"
items[2] = "B"
arraypop(items) == "B"
items[0] == 1
```

The array is intentionally exposed because mutation is the operation's contract. The implementation reads the high-water mark once and uses one native `DELATTRS1` deletion at the known-valid last index; empty input returns before that instruction. It performs no scan, shift, helper call, or allocation under Level B control. There is no Level C BIF or class method named ARRAYPOP.

`lib/rxfnsb/tests_functional/ts_arraypop.crexx` covers values, empty-string elements, repeated mutation, explicit/omitted defaults, and high-water state.

7.24 arrayprepend

```
arrayprepend(array = .string[] expose,
             value = .string,
             count = .int optional) = .int
```

`arrayprepend` adds `value` to the front of `array` `count` times and returns the new high-water mark. `count` defaults to one. The array uses `array[0]` as its automatically maintained high-water mark.

A zero count is a no-op. A negative count raises `INVALID_ARGUMENTS`.

The exposed array is mutated in place. The implementation uses one VM `insattrs1` bulk shift followed by exactly one value assignment per new slot; it does not move the existing tail in REXX code. The focused harness is `lib/rxfnsb/tests_functional/ts_arrayprepend.crexx`.

7.25 arrayreverse

`arrayreverse` reverses a one-based Level B string array in place:

```
arrayreverse(expose array=.string[]) = .int
```

The return value is the unchanged high-water mark. Empty, single-element, empty-string, and Unicode elements are ordinary array values.

```
items = .string[]
items[1] = "A"
items[2] = "B"
items[3] = "C"
arrayreverse(items) == 3
items[1] == "C"
```

The implementation swaps the first/last, second/penultimate, and subsequent pairs until the indices meet. It is O(n), uses constant temporary storage, and performs exactly half as many swaps as elements; it does not allocate or return a copied array and has no helper calls. There are no invalid-input branches, Level C BIF, or class method named ARRAYREVERSE.

`lib/rxfnsb/tests_functional/ts_arrayreverse.crexx` covers empty, single, odd, even, Unicode/empty values, return state, and reversing twice.

7.26 arrayset

```
arrayset(array = .string[] expose, index = .int, value = .string
         [, fill = .string]) = .int
```

`arrayset` assigns `value` to the positive one-based `index` and returns the resulting `array[0]` high-water mark. An existing element is overwritten in place. When `index` is beyond the current end, the array grows through that index and each intervening slot receives `fill`; the omitted fill is the empty string.

A non-positive `index` signals `INVALID_ARGUMENTS`. The array is exposed and mutated in place. Growth uses one bulk attribute insertion, skips explicit gap initialisation for an empty fill, and never writes `fill` into the target slot before overwriting it with `value`.

The focused harness is `lib/rxfnsb/tests_functional/ts_arrayset.crexx`.

7.27 arrayshift

`arrayshift` removes the first element of a one-based Level B string array:

```
arrayshift(expose array=.string[] [,default=.string]) = .string
```

When the array is nonempty, its first value is returned, later elements move down one index, and its high-water mark is reduced by one. Empty-string elements are values and are still removed. Empty input is unchanged and returns `default`, whose default is the empty string.

```
items = .string[]
items[1] = "A"
items[2] = "B"
arrayshift(items) == "A"
items[1] == "B"
```

The exposed array is intentional. The implementation reads the first value and uses one native `DELATTRS1` deletion; that instruction performs the required O(n) attribute shift without a Level B loop or helper call. Empty input returns before mutation. There is no Level C BIF or class method named ARRAYSHIFT.

`lib/rxfnsb/tests_functional/ts_arrayshift.crexx` covers shifted indices, empty-string/Unicode values, repeated mutation, defaults, and high-water state.

7.28 arraysort

`arraysort` sorts a one-based Level B string array in place:

```
arraysort(expose array=.string[] [,offset=.int [,order=.string [,debug
    =.int]]]) = .int
```

`offset` is the positive one-based character position at which each lexical, case-sensitive key begins and defaults to `1`. Text before it remains part of the moved value but not the comparison. A position beyond a value yields an empty key. `order` is case-insensitive `ASC` (default) or `DESC`. `debug` is `0` by default; `1` prints one summary line. The return is the unchanged high-water mark. Equal-key order is not stable.

Invalid offset, order, or debug values signal INVALID_ARGUMENTS. Because the array is exposed, caught signals currently share the repository's parked exposed-argument unwind dependency; run each error assertion in a fresh VM.

The implementation extracts every substring key once, then moves values and keys together through an in-place binary heap. Runtime is O(n log n), auxiliary key storage is O(n), and comparisons allocate no substrings. It performs no general selector calls in the sort core. There is no Level C BIF or class method named ARRAYSORT.

ts_arraysort.crexx covers valid orders, substring keys, lexical numeric text, Unicode, empty/single arrays, and return state. ts_arraysort_errors.crexx covers each signal in a separate process.

7.29 b2d

```
b2d(binary = .string) = .int
```

b2d converts binary digit text to a non-negative Level B integer. The empty string and strings containing only zero digits return 0.

Interior ASCII blanks may separate groups only when a multiple of four binary digits appears to their right. Leading/trailing blanks, misplaced blanks, and characters other than 0, 1, and blank signal INVALID_ARGUMENTS.

```
b2d("01110")       /* 14 */
b2d("10000001")    /* 129 */
b2d("10 1010")     /* 42 */
```

The return type is the signed 64-bit .int, so the greatest accepted unsigned result is 9223372036854775807 (63 one bits). A larger significant value signals OVERFLOW_UNDERFLOW; any number of leading zero bits is harmless.

This is a Level B native helper, not a repository Level C BIF. It has one argument and does not implement the optional signed-width extension described by older imported class-reference material.

7.30 b2x

```
b2x(binary = .string) = .string
```

b2x converts binary digit text to uppercase hexadecimal text. The empty string is valid and returns the empty string. A partial leading group produces one hexadecimal digit; every complete four-bit group produces one further digit, including zero nibbles.

Interior ASCII blanks may separate groups only when a multiple of four binary digits appears to their right. Leading/trailing blanks, misplaced blanks, and characters other than 0, 1, and blank signal INVALID_ARGUMENTS.

```
b2x("11000011")  /* C3 */
b2x("10111")     /* 17 */
b2x("10 1010")   /* 2A */
b2x("")          /* empty */
```

The implementation scans the input directly and emits one nibble at a time. It does not convert the complete bit string through an integer, so input length is not constrained by integer or decimal precision.

Classic Level C callers use the RexxValue/error-context contract documented in lib/rxfnsc/b2x.md.

7.31 Level B binary helpers

binary.crexx is the typed Level B binary library. It has two deliberately different position conventions:

- the copy-returning BIN* value helpers use 1-based byte positions;
- the exposed packed-memory mutators use zero-based byte offsets matching <at..type> and RXAS binary-memory instructions.

This module is not a Level C Classic BIF. There is consequently no same-named lib/rxfnsc implementation or Level C contract.

7.32 Binary value helpers

Function	Result	Contract
binlength(data = .binary)	.int	Logical byte length.
binbyte(data = .binary, position = .int)	.int	Byte at a 1-based position, or -1 outside the value.

Function	Result	Contract
`binsetbyte(data = .binary, position = .int, byte = .int)`	`.binary`	Copy with one byte replaced.
`binsubstr(data = .binary, start = .int, count = -1)`	`.binary`	Copied slice; a negative count means through the end.
`binconcat(left = .binary, right = .binary)`	`.binary`	Byte concatenation.
`binoverlay(new = .binary, target = .binary, start = .int)`	`.binary`	Fixed-size overlay into a copy.
`bininsert(new = .binary, target = .binary, before = .int)`	`.binary`	Insert before a 1-based position; prepend at `before <= 1`, append past the end.
`bindelstr(target = .binary, start = .int, count = 0)`	`.binary`	Delete bytes; zero means through the end.
`binpos(needle = .binary, haystack = .binary, start = 1)`	`.int`	First 1-based match or zero; an empty needle returns zero.
`bincompare(left = .binary, right = .binary)`	`.int`	Zero when equal, otherwise the first differing 1-based position.
`bin2x(data = .binary)`	`.string`	Two uppercase hexadecimal digits per byte.
`x2bin(hex = .string)`	`.binary`	Decode hex, ignoring ASCII blanks and left-padding an odd nibble.

`binsetbyte` and `binoverlay` raise `OUT_OF_RANGE` for invalid byte ranges; `binsetbyte` also uses it for a byte outside `0..255`. `binsubstr`, `bindelstr`, and `binpos` raise `INVALID_ARGUMENTS` for a nonpositive 1-based start. `bindelstr` raises it for a negative count, and `x2bin` raises it for any non-blank character that is not hexadecimal.

```
data = "001122ff"x as .binary
binlength(data)                                  /* 4 */
binbyte(data, 2)                                 /* 17 */
bin2x(binsubstr(data, 2, 2))                     /* "1122" */
bin2x(binoverlay("abcd"x as .binary, data, 2))  /* "00ABCDFF" */
binpos("1122"x as .binary, data)                /* 2 */
bincompare(data, "001123ff"x as .binary)        /* 3 */
bin2x(x2bin("ff 00 aa"))                        /* "FF00AA" */
```

The search implementation compares each candidate directly in the binary buffer; it does not materialize a slice or call a byte helper for every byte. Equal-length bincompare values also take the whole-value zero-copy fast path.

7.33 Packed-memory mutators

The first argument of each mutator is declared with `arg expose`, so the caller's binary variable changes in place. Successful calls return the new logical byte length, except that fixed-size operations naturally return the unchanged length.

Function	Effect
`binresize(data, length)`	Preserve or zero-grow to `length`.
`binclear(data)`	Set the logical length to zero.
`binfill(data, byte)`	Fill every current byte.
`binfillat(data, offset, length, byte)`	Fill a zero-based span.
`bincopy(dst, dst_offset, src, src_offset, length)`	Copy between binary values.
`binmemmove(data, dst_offset, src_offset, length)`	Overlap-safe move within one value.
`binappend(dst, src)`	Append every source byte.
`binupdate(dst, offset, src)`	Fixed-size overlay.
`binmakegap(data, offset, length)`	Open a zero-filled gap.
`bindrop(data, offset, length)`	Delete a span.

Every offset and length is `.int`. Invalid spans, negative lengths, and bytes outside `0..255` raise `OUT_OF_RANGE`. A zero-length span is valid when its offset is from zero through the current length, inclusive. Fixed-size writes never grow the destination; use `binresize`, `binappend`, or `binmakegap` when growth is required.

```
page = "00112233"x as .binary
call binmakegap page, 2, 2       /* 00 11 00 00 22 33 */
call binfillat page, 2, 2, 255   /* 00 11 FF FF 22 33 */
call bindrop page, 2, 2          /* 00 11 22 33 */
```

`bincopy` and `binmemmove` lower inside the library to the VM's checked move instructions, avoiding a temporary chunk. `binmakegap` and `bindrop` likewise use overlap-safe in-buffer moves.

The focused harness is `lib/rxfnsb/tests_functional/ts_binary.crexx`. It covers the examples, copy-versus-mutation behavior, long search, overlap in both directions, zero-length boundary spans, and each documented signal family in optimized and unoptimized selector overlays.

7.34 c2d

```
c2d(from = .string) = .int
```

The Level B `c2d` helper returns the native integer Unicode code point of exactly one character. Its typed integer result is suitable for comparisons, indexes, byte/code-point classification, and `d2c` round trips without a string-to-number conversion.

```
c2d("M")    /* 77 */
c2d("□")    /* 945 */
c2d("□")   /* 128293 */
c2d("00"x) /* 0 */
```

An empty or multi-character string raises `CONVERSION_ERROR`. The implementation computes the Unicode character length once and extracts the single code point directly.

This is not the Classic Level C C2D contract. Level C accepts an optional length and interprets configuration-coded characters as a signed twos-complement value. That separate contract is documented in `lib/rxfnsc/c2d.md`.

The focused optimized/unoptimized harness is `lib/rxfnsb/tests_functional/ts_-c2d.crexx`.

7.35 c2x

```
c2x(from = .string) = .string
```

`c2x` converts every source character to two uppercase hexadecimal digits. The empty string returns the empty string; multi-character strings produce two digits per character and leading zero digits are retained.

```
c2x("M")        /* 4D */
c2x("72s")      /* 373273 on the ASCII/Unicode build */
c2x("0123"x)   /* 0123 */
c2x("")         /* empty */
```

The current Level B contract preserves the established RXAS `hexchar` behavior: for a Unicode code point above the single-byte range it formats the low eight bits. For example, `c2x("é")` is `E9`. Changing C2X to emit the underlying UTF-8 bytes would be a separate language-surface decision and is not part of this library-only implementation change.

The input is a typed `.string`, so there is no value-domain error. The implementation scans once and appends each two-digit group directly rather than re-concatenating the growing result.

The focused harness is `lib/rxfnsb/tests_functional/ts_c2x.crexx`. The separate Classic Level C contract is in `lib/rxfnsc/c2x.md`.

7.36 center

`center` pads or centrally truncates text to an exact character width:

```
center(string = .string, width = .int, pad = " ") = .string
```

- `width` must be non-negative.
- `pad` must contain exactly one Unicode codepoint.
- When padding is uneven, the extra pad character is placed on the right.
- When truncation is uneven, the extra removed character comes from the right; equivalently, the retained central slice is biased one character left.

Invalid width or pad input raises `INVALID_ARGUMENTS`.

```
center("ABC", 7)            /* "  ABC  " */
center("ABC", 8, "-")       /* "--ABC---" */
center("The blue sky", 8)   /* "e blue s" */
center("anything", 0)       /* "" */
```

The source argument is read-only. The implementation computes character lengths once, uses VM cursor/subslice operations for truncation, and creates one half-padding string which is appended on both sides.

`lib/rxfnsb/tests_functional/ts_center.crexx` is CENTER-only; the separate CENTRE alias is reviewed and tested in its own selector row.

7.37 centre

`centre` is the British-spelling Level B entry for exact-width centring and central truncation:

```
centre(string = .string, width = .int, pad = " ") = .string
```

- `width` must be non-negative.

- `pad` must contain exactly one Unicode codepoint.
- An uneven extra pad character is placed on the right.
- Uneven truncation removes the extra character from the right, retaining the left-biased central slice.

Invalid width or pad input raises `INVALID_ARGUMENTS`.

```
centre("ABC", 7)            /* "  ABC  " */
centre("ABC", 8, "-")       /* "--ABC---" */
centre("The blue sky", 8)   /* "e blue s" */
centre("anything", 0)       /* "" */
```

The source argument is read-only. This performance-critical Level B entry uses the same reviewed direct VM algorithm as `center`: lengths are computed once, truncation uses cursor/subslice operations, and one half-padding allocation is reused on both sides.

`lib/rxfnsb/tests_functional/ts_centre.crexx` is CENTRE-only. The American spelling has separate source, tests, and documentation.

7.38 changestr

`changestr` replaces every case-sensitive, non-overlapping occurrence of `needle` in the original `haystack`:

```
changestr(needle = .string, haystack = .string,
          replacement = .string) = .string
```

Searching proceeds from left to right. Inserted replacement text is never searched again, even when it contains `needle`. A null `needle` or a needle that is not found returns the haystack unchanged. A null replacement deletes matches.

```
changestr("the", "the cat and the dog", "a")  /* "a cat and a dog" */
changestr("aa", "aaaaa", "X")                 /* "XXa" */
changestr("a", "banana", "")                 /* "bnn" */
changestr("", "unchanged", "!")              /* "unchanged" */
```

All inputs are read-only and there are no value-domain signals. Their static Level B types enforce the three mandatory string arguments.

The implementation searches the original haystack directly and appends each unmatched slice and replacement to one result. It does not rebuild and re-search the growing result after every match.

`lib/rxfnsb/tests_functional/ts_changestr.crexx` covers deletion, expansion, non-overlap, empty inputs, case sensitivity, Unicode, argument non-mutation, and a repeated replacement workload in optimized and unoptimized overlays.

7.39 compare

`compare` returns the first 1-based character position at which two strings differ after the shorter string is virtually padded:

```
compare(left = .string, right = .string, pad = " ") = .int
```

The result is zero when the strings compare equal. `pad` must contain exactly one Unicode codepoint; an invalid pad raises `INVALID_ARGUMENTS`.

```
compare("abc", "abc")        /* 0 */
compare("abc", "ak")         /* 2 */
compare("ab ", "ab")         /* 0: default blank padding */
compare("ab-- ", "ab", "-")  /* 5 */
```

Both source strings are read-only. The implementation compares their common prefix directly by codepoint and compares only the longer tail with the cached pad codepoint. It allocates no padded copy of either string.

`lib/rxfnsb/tests_functional/ts_compare.crexx` covers equal and differing strings, both longer-side directions, blank/nonblank/Unicode padding, Unicode positions, invalid-pad signals, and non-mutation in optimized and unoptimized overlays.

7.40 copies

`copies` concatenates a string a non-negative number of times:

```
copies(string = .string, count = .int) = .string
```

Count zero returns the null string; count one returns the source value. A negative count raises `INVALID_ARGUMENTS`.

```
copies("abc", 3)  /* "abcabcabc" */
copies("abc", 0)  /* "" */
copies("", 2)     /* "" */
```

The source argument is read-only. For larger counts the implementation uses binary decomposition: it doubles a repeated chunk and appends only chunks for set bits in `count`. This reduces the number of VM append operations from `count` to O(log `count`) while producing the same exact result.

`lib/rxfnsb/tests_functional/tscopies.crexx` covers zero, one, ordinary, non-power-of-two, Unicode, empty, negative, and 4,096-copy cases in optimized and unoptimized overlays.

7.41 countstr

`countstr` counts case-sensitive, non-overlapping occurrences from left to right:

```
countstr(needle = .string, haystack = .string) = .int
```

A null needle, empty haystack, oversized needle, or absent needle returns zero.

```
countstr("bc", "abcabcabc")  /* 3 */
countstr("aa", "aaaaa")      /* 2, not 4 */
countstr("", "anything")     /* 0 */
```

Both inputs are read-only and there are no value-domain signals. The implementation computes lengths once and performs one direct VM substring search per match, advancing by the full needle length.

`lib/rxfnsb/tests_functional/ts_countstr.crexx` covers empty, absent, oversized, boundary, non-overlapping, Unicode, non-mutation, and 1,000-match cases in optimized and unoptimized overlays.

7.42 d2b

```
d2b(decimal = .int) = .string
```

`d2b` converts a non-negative native integer to minimal binary digit text. There are no leading zeroes, except that zero itself returns `"0"`.

```
d2b(0)   /* "0" */
d2b(9)   /* "1001" */
d2b(19)  /* "10011" */
d2b(129) /* "10000001" */
```

A negative input raises `INVALID_ARGUMENTS`. This native helper has no width argument and therefore has no signed twos-complement form. Callers needing a fixed-width encoded representation should use the appropriate typed binary memory operation rather than relying on implicit truncation.

The implementation counts the significant bits with logical shifts, then emits them directly from most significant to least significant. It does not call `d2x` or `x2b`, allocate an intermediate representation, or inherit those functions' grouping and diagnostic contracts.

There is no same-named Classic Level C BIF in the repository recognition ledger. The focused optimized/unoptimized harness is `lib/rxfnsb/tests_functional/ts_d2b.crexx`.

7.43 d2c

```
d2c(codepoint = .int [, output_length = .int]) = .string
```

The Level B `d2c` helper converts a native Unicode scalar value to one character. Valid scalar values are `0` through `1114111` (`U+10FFFF`), except the surrogate range `55296` through `57343` (`U+D800` through `U+DFFF`).

```
d2c(77)       /* "M" */
d2c(945)      /* "□" */
d2c(128293)   /* "□" */
d2c(0)        /* NUL */
```

When `output_length` is omitted or one, one character is returned. An explicit zero returns the empty string without encoding the integer. Any other explicit length, or an invalid scalar value when a character is requested, raises `CONVERSION_ERROR`.

The arguments are native `.int` values. The implementation performs bounded scalar validation and one `appendchar`; it does not convert through hexadecimal or call another selector.

This is not the Classic Level C D2C contract. Level C converts a whole-number RexxValue to configuration-coded characters, supports signed twos-complement when a length is present, and pads or truncates in encoded-character units. That separate contract is documented in `lib/rxfnsc/d2c.md`.

The focused optimized/unoptimized harness is `lib/rxfnsb/tests_functional/ts_d2c.crexx`.

7.44 d2x

```
d2x(number = .int [, output_length = .int]) = .string
```

The Level B `d2x` helper converts a native signed 64-bit integer to uppercase hexadecimal digit text. Without `output_length`, `number` must be non-negative and the result uses the minimum width (`d2x(0)` is `"0"`).

With a non-negative `output_length`, the result has exactly that many digits. Positive values are padded with `0`, negative values are represented in twos-complement and padded with `F`, and excess high digits are discarded.

```
d2x(129)        /* "81" */
d2x(129, 4)     /* "0081" */
d2x(257, 2)     /* "01" */
d2x(-127, 2)    /* "81" */
d2x(-127, 4)    /* "FF81" */
d2x(12, 0)      /* empty */
```

A negative value without a supplied width or a negative width raises `INVALID_ARGUMENTS`. Both arguments are native `.int` values.

The implementation extracts the requested nibbles directly from the native integer. It performs at most 16 nibble extractions plus one bulk left pad and does not call `right`, divide per digit, or build and reverse an intermediate string.

The separate Classic Level C BIF accepts caller-context Rexx whole numbers and is documented in `lib/rxfnsc/d2x.md`. The focused native harness is `lib/rxfnsb/tests_functional/ts_d2x.crexx`.

7.45 datatype

```
datatype(value = .string [, type = .string]) = .string
```

With `type` omitted, `DATATYPE` returns `NUM` for a syntactically valid Level B number and `CHAR` otherwise. An explicit type tests the first option codepoint and returns `1` or `0`.

Level B recognizes `A`, `B`, `D`, `L`, `M`, `N`, `S`, `U`, `W`, and `X`. `D` is the cRexx extension for ASCII digits. The other character classes are also deliberately ASCII-based; traversal is codepoint-safe, but this foundation function does not claim full Unicode property behavior.

`B` and `X` accept the empty string. Embedded ASCII spaces are allowed only at right-counted group boundaries: multiples of four binary digits and pairs of hexadecimal digits. Leading and trailing spaces are invalid. Other non-numeric classes reject the empty string.

`N` supports surrounding blanks, an optional sign, decimal point, and signed decimal exponent. `W` applies the same syntax and tests the exact mathematical value. It does not convert through binary floating point or use a tolerance, so `123.000` is whole while `123.0000003` is not.

An explicit empty or unsupported option signals `INVALID_ARGUMENTS`. The input is never modified.

The separate Classic Level C BIF is documented in `lib/rxfnsc/datatype.md`; it intentionally excludes the Level B `D` extension.

7.46 date

The typed Level B calendar extension is:

```
date([oformat="" [,idate="" [,iformat="" [,osep="" [,isep=""]]]]]) = .
    string
```

An empty `idate` uses the current local date and ignores `iformat`. The default format is `NORMAL`. Calendar years are the proleptic Gregorian range 1 through 9999. Every result, including BASE, UNIX, EPOCH, and JDN, is a `.string`.

This is deliberately distinct from the three-argument Classic Level C DATE BIF documented in `lib/rxfnsc/date.md`.

7.46.1 Formats

Format names are case-insensitive documented abbreviations. X variants carry a four-digit year; the corresponding short numeric input variants map `00` through `99` deterministically to 2000 through 2099.

Format	Output example for 1962-03-10	Input
`NORMAL` / `N`	`10 Mar 1962`	yes
`XNORMAL` / `XN`	`10 March 1962`	yes
`STANDARD` / `ST`	`19620310`	yes

Format	Output example for 1962-03-10	Input
`ORDERED` / `O`	`62/03/10`	yes, interpreted as 2062
`XORDERED` / `XO`	`1962/03/10`	yes
`EUROPEAN` / `E`	`10/03/62`	yes, interpreted as 2062
`XEUROPEAN` / `XE`	`10/03/1962`	yes
`GERMAN` / `G`	`10.03.62`	yes, interpreted as 2062
`XGERMAN` / `XG`	`10.03.1962`	yes
`USA` / `U`	`03/10/62`	yes, interpreted as 2062
`XUSA` / `XU`	`03/10/1962`	yes
`INTERNATIONAL` / `IN`	`1962-03-10`	yes
`QUALIFIED` / `QU`	`Saturday, March 10, 1962`	yes; weekday is validated
`JULIAN` / `J`	`1962069`	yes
`BASE` / `B`	`716308`	yes; days since 0001-01-01
`UNIX` / `UN`	`-2854`	yes; days since 1970-01-01
`EPOCH` / `EP`	`-246585600`	yes; seconds since 1970-01-01
`JDN`	`2437734`	yes
`DAYS` / `D`	`069`	output only
`WEEKDAY` / `W`	`Saturday`	output only
`MONTH` / `M`	`March`	output only
`CENTURY` / `C`	`22714`	output only
`DEC`	`10-03-62`	yes, interpreted as 2062
`XDEC`	`10-03-1962`	yes
`SORTED` / `S`	`19620310`	output only

`S` means SORTED on output and STANDARD on input. Use `ST` when naming STANDARD output explicitly.

7.46.2 Separators and errors

`osep` and `isep` are empty or exactly one Unicode codepoint. A supplied separator replaces the format's normal separator; SORTED normally has none. For example:

```
date("XG", "••10031962", "XE", "•", "•") -- ••10031962
```

Invalid dates, unknown or output-only input formats, malformed fields, range errors, and overlong separators signal `INVALID_ARGUMENTS`. Inputs are scanned directly and do not print diagnostics or normalize invalid calendar fields.

`ts_date`, `ts_date2`, and `ts_date_core` exercise every table row, the Unicode example, Gregorian boundaries, invalid inputs, and optimized/unoptimized code.

7.47 delstr

`delstr` removes a character range from a string:

```
delstr(string = .string, start = .int,
       delete_length = 0) = .string
```

`start` is a positive 1-based Unicode character position. When `delete_length` is omitted, deletion continues through the end. An explicitly supplied zero deletes nothing. A start beyond the string also returns the source unchanged.

Nonpositive `start` or a negative supplied `delete_length` raises `INVALID_ARGUMENTS`.

```
delstr("abcd", 3)      /* "ab" */
delstr("abcde", 3, 2) /* "abe" */
delstr("abcde", 6)    /* "abcde" */
delstr("abcde", 3, 0) /* "abcde" */
```

The source is read-only. The implementation distinguishes omission through argument-presence metadata, computes length once, and extracts at most one prefix and one suffix with direct VM cursor/subslice operations.

`lib/rxfnsb/tests_functional/ts_delstr.crexx` covers omitted/zero/oversized lengths, boundary starts, empty/Unicode input, signals, and non-mutation in optimized and unoptimized overlays.

7.48 delword

```
delword(string = .string, start = .int [, count = .int]) = .string
```

`delword` returns `string` with words beginning at the positive one-based `start` removed. An omitted `count` deletes through the final word; a supplied zero returns the source unchanged, and a count larger than the available words clamps to the remaining words. A start beyond the word count also returns the source unchanged.

Deletion includes all whitespace following the final deleted word but none of the whitespace preceding the first deleted word. Level B uses the VM Unicode whitespace definition. A non-positive start or supplied negative count signals `INVALID_ARGUMENTS`.

The implementation scans the source once with direct blank/nonblank VM operations and composes at most one prefix and one tail; it makes no other selector calls. The focused harness is `lib/rxfnsb/tests_functional/ts_delword.crexx`.

7.49 filter

`filter` removes characters from a string:

```
filter(input_value=.string, filter=.string) = .string
```

Every source character that occurs in `filter` is removed. Comparison is by Unicode code point, repeated removal characters have no additional effect, and the retained characters keep their original order.

```
filter("□□□abc", "□□") == "abc"
```

An empty removal set returns an equal copy of the source; an empty source returns empty. Both arguments remain unchanged. Valid Level B `.string` values require valid UTF-8; the underlying VM instruction raises `UNICODE_ERROR` if an invalid string reaches it, and allocation failure is fatal.

The implementation initializes one result and executes the VM `dropchar` instruction once. The VM compares the removal set directly without a Level B helper call or intermediate normalized copy. There is no Level C BIF or class method with this name. `lib/rxfnsb/tests_functional/ts_filter.crexx` covers the documented behavior.

7.50 find

`find` is the VM/TSO-compatible argument-order alias of Level B `wordpos`:

```
find(needle=.string, haystack=.string [,start=.int]) = .int
```

It returns the one-based word number at which the exact, case-sensitive word or phrase first occurs, or zero. `start` is a one-based word number and defaults to `1`. Runs of Unicode whitespace separate words. Prefixes are not matches.

```
find("the", "Now is the time") == 3
find("be", "To be or not to be", 3) == 6
```

An empty/blank phrase or haystack returns zero. A non-positive `start` signals `INVALID_ARGUMENTS`. Inputs are not modified.

The implementation deliberately delegates to the reviewed Level B `wordpos` algorithm so the compatibility alias cannot drift from its word semantics. It adds one fixed function call and no string copy or scan of its own. There is no Level C

BIF or class method named FIND. The focused harness is `lib/rxfnsb/tests_-functional/ts_find.crexx`.

7.51 floatabs

```
floatabs(number = .float) = .float
```

Returns the binary64 absolute value without decimal conversion. Positive and negative zero both return positive zero, either infinity returns positive infinity, and NaN propagates. NaN payload/sign details are not part of the public contract.

Invalid dynamic conversion to `.float` signals `CONVERSION_ERROR`.

7.52 floatformat

```
floatformat(number = .float,
            before = .int optional,
            after = .int optional,
            expp = .int optional,
            expt = .int optional) = .string
```

Formats a finite binary64 value using the same omission, rounding, width, exponent-trigger, `NUMERIC DIGITS`, `NUMERIC FORM`, and `NUMERIC CASE` rules as decimal `format`. The value is extracted directly to numeric-context text and is never converted to decimal.

Signed zero is canonicalized by native extraction. Non-finite values, negative controls, and fields that cannot fit signal `INVALID_ARGUMENTS`. Typed conversion failures signal `CONVERSION_ERROR`. The result is ordinary `.string` text.

7.53 floatmax

```
floatmax(first = .float, ... = .float) = .float
```

Returns the greatest of one or more homogeneous binary64 values without decimal conversion. Infinities are ordered normally. The first equal value is retained, including the sign of equal zero values. Any NaN signals `INVALID_ARGUMENTS`.

Invalid dynamic argument conversion signals `CONVERSION_ERROR`.

7.54 floatmin

```
floatmin(first = .float, ... = .float) = .float
```

Returns the least of one or more homogeneous binary64 values without decimal conversion. Infinities are ordered normally. The first equal value is retained, including the sign of equal zero values. Any NaN signals `INVALID_ARGUMENTS`.

Invalid dynamic argument conversion signals `CONVERSION_ERROR`.

7.55 floatsign

```
floatsign(number = .float) = .int
```

Returns `-1`, `0`, or `1` after binary64 comparisons. Both signed zeros return zero, infinities return their respective sign, and NaN signals `INVALID_ARGUMENTS`. No decimal payload is created.

Invalid dynamic conversion to `.float` signals `CONVERSION_ERROR`.

7.56 floattrunc

```
floattrunc(number = .float, fraction = .int optional) = .string
```

`fraction` defaults to zero. The finite binary64 value is extracted under the caller's current `NUMERIC DIGITS` and `NUMERIC CASE`, then truncated toward zero to fixed non-exponent text with exactly `fraction` fractional digits. Extraction is native and never creates a decimal payload.

Exact signed zero becomes unsigned zero text. A negative nonzero value that truncates to zero retains its minus sign. Negative `fraction` and non-finite inputs signal `INVALID_ARGUMENTS`; typed conversion failures signal `CONVERSION_ERROR`. The result is ordinary `.string` text and receives no hidden numeric return type.

7.57 fnv

The typed Level B function returns the conventional 32-bit FNV-1a hash of a text value:

```
fnv(input_value = .string) = .int
```

The input is hashed as its exact UTF-8 byte sequence, in forward byte order. The result is the unsigned 32-bit value represented as an integer in `0..4294967295`.

For example, `fnv("")` is `2166136261`, `fnv("a")` is `3826002220`, and `fnv("abc")` is `440920331`.

Embedded NUL bytes participate in the hash. The function does not mutate the input and does not signal numeric overflow because the algorithm deliberately uses modulo-2^32 unsigned arithmetic. Invalid text is rejected by the normal `.string`/UTF-8 runtime rules before it can be hashed.

`fnv` is a Level B library function, not a Level C Classic BIF. The backing `rxhash` instruction accepts an explicit byte count; `fnv` supplies the full encoded byte length.

7.58 format

```
format(number = .decimal,
       before = .int optional,
       after = .int optional,
       expp = .int optional,
       expt = .int optional) = .string
```

`format` is the native Level B decimal formatter. `number` is converted to a decimal at the typed call boundary; all four formatting controls are integers. The function inherits the caller's `NUMERIC DIGITS` and `NUMERIC FORM` settings.

- `before` is the width of the integer part, including a possible minus sign.
- `after` is the exact number of fractional digits; excess digits are rounded half up and missing digits are zero-filled.
- `expp` is the exponent digit width. A supplied zero suppresses exponential notation; a supplied width leaves a blank exponent field when the exponent is zero.
- `expt` is the trigger for exponential notation. When it is omitted but `expp` is present, the current `NUMERIC DIGITS` value is used.

Omission is significant. For example, `format(value,,4)` leaves `before` unspecified while requesting four fractional digits.

```
format(1.75, 4, 1)              /* "   1.8" */
format(12345.73,,,2,2)          /* "1.234573E+04" */

numeric form engineering
format(12345,,3,3,0)            /* "12.345E+003" */
```

Negative controls and integer/exponent fields that cannot fit signal `INVALID_ARGUMENTS`. Invalid dynamic conversion to the typed `.decimal` or `.int` parameters follows the Level B conversion-signal contract.

The implementation performs decimal-preserving string decomposition and exact digit rounding. It does not pass through binary floating point and does not call the general Level B string BIFs on the optimized path.

Classic Level C callers use the distinct RexxValue contract documented in `lib/rxfnsc/format.md`.

7.59 fsayfmt

`fsayfmt` converts an FSAY template into a CREXX source expression:

```
fsayfmt(template = .string) = .string
```

It is a source generator, not a runtime interpolation engine. Placeholders may name a simple identifier or compound variable. General expressions and array subscripts are rejected. The compiler-exit statement passes one quoted source literal; direct calls normally pass the decoded template text.

```
fsayfmt("Hello {name}")         /* 'Hello '||name */
fsayfmt("{name:10}")           /* left(name,10) */
fsayfmt("{price:8.2}")         /* format(price,8,2) */
```

The placeholder grammar is `{variable[:alignment][width][.decimals]}`. Whitespace may replace the colon. Unqualified text widths are left aligned. `<`, `>` and `^` select left, right and centre alignment. Decimal formatting uses `FORMAT`; its default is right alignment. A decimal precision may be supplied without a width as `.digits`. Widths and decimal counts are non-negative decimal integers.

`{{` and `}}` emit literal braces. Quote characters inside the decoded template are ordinary literal text. Generated literal segments always use single-quoted CREXX source with embedded apostrophes doubled.

Malformed braces, placeholders, variable names, widths or decimal counts signal `INVALID_ARGUMENTS`. The implementation decodes at most one outer source literal and scans the resulting template once; it does not invoke the general quote-aware library or rescan the growing output.

`lib/rxfnsb/tests_functional/ts_fsayfmt.crexx` covers source-literal decoding, literal escaping, braces, compound names, all alignment modes, decimal formats, Unicode text and each documented validation failure in optimized and unoptimized Level B builds. The compiler-exit integration test additionally executes the generated FSAY statements.

7.60 getenv

`getenv` reads one variable from the running process environment:

```
getenv(env_name=.string) = .string
```

The name is passed directly to the host platform. The result is the variable's value, or the empty string when the variable is missing. A missing variable and a variable whose value is empty are intentionally indistinguishable.

```
home = getenv("HOME")
```

The input is not modified. The implementation executes the VM `getenv` instruction once and has no Level B helper calls or intermediate string copy. There is no Level C BIF with this name and no library signal contract; allocation failure in the VM is fatal.

`lib/rxfnsb/tests_functional/ts_getenv.crexx` checks an injected value containing spaces, repeatability, missing and empty names, and input non-mutation.

7.61 index

`index` is the VM/TSO-compatible haystack-first spelling of substring search:

```
index(haystack=.string, needle=.string [,start=.int]) = .int
```

It returns the one-based Unicode character position of the first exact, case-sensitive occurrence at or after `start`, or zero. `start` defaults to `1`.

```
index("Saturday", "day") == 6
index("banana", "ana", 3) == 4
```

An empty needle or haystack returns zero. A non-positive `start` signals `INVALID_ARGUMENTS`. Inputs are read-only.

The implementation validates once and executes the VM `strpos` instruction directly. It performs no substring allocation, text copy, or call through POS; only the public argument order differs. There is no Level C BIF or class method named INDEX. `lib/rxfnsb/tests_functional/ts_index.crexx` covers the documented surface.

7.62 insert

`insert` adds formatted text after a character position in a target string:

```
insert(new_string = .string, target = .string, before = 0,
       insert_length = 0, pad = " ") = .string
```

`before` and `insert_length` are optional non-negative integers. The default position is zero, before the first target character. When `insert_length` is omitted it is the character length of `new_string`; an explicitly supplied zero inserts no new text. `pad` defaults to blank and must contain exactly one Unicode codepoint.

If `before` is beyond the target, padding extends the target up to the insertion point. The inserted text is truncated or padded to `insert_length`.

Invalid position, length, or pad input raises `INVALID_ARGUMENTS`.

```
insert("123", "abc")              /* "123abc" */
insert(" ", "abcdef", 3)          /* "abc def" */
insert("123", "abc", 5, 6, "+")  /* "abc++123+++" */
insert("abc", "def", 2, 1)       /* "deaf" */
```

Arguments are read-only. The implementation caches character lengths and uses direct VM substring, append, and pad operations. It does not call `LEFT`, `SUBSTR`, `COPIES`, or another Level B formatting helper.

7.63 intabs

```
intabs(number = .int) = .int
```

Returns the native signed-integer absolute value without a decimal or float conversion. Zero and positive values are returned unchanged. The absolute value of `INT64_MIN` is not representable and signals `OVERFLOW_UNDERFLOW`.

Invalid dynamic conversion to `.int` follows the normal Level B `CONVERSION_ERROR` contract.

7.64 intformat

```
intformat(number = .int,
          before = .int optional,
          after = .int optional,
          expp = .int optional,
          expt = .int optional) = .string
```

Formats the full native signed-64-bit input range using the same omission, rounding, width, exponent-trigger, `NUMERIC DIGITS`, `NUMERIC FORM`, and `NUMERIC CASE` rules as decimal `format`. The integer is extracted directly to numeric-context text and is never converted to decimal or float.

Negative controls and fields that cannot fit signal `INVALID_ARGUMENTS`. Typed conversion failures signal `CONVERSION_ERROR`. The result is ordinary `.string` text.

7.65 intmax

```
intmax(first = .int, ... = .int) = .int
```

Returns the greatest of one or more native signed integers. All variadic arguments are homogeneous `.int` values, comparison is native, and the first occurrence of an equal maximum is retained. There is no arithmetic overflow path.

Invalid dynamic argument conversion signals `CONVERSION_ERROR`.

7.66 intmin

```
intmin(first = .int, ... = .int) = .int
```

Returns the least of one or more native signed integers. All variadic arguments are homogeneous `.int` values, comparison is native, and the first occurrence of an equal minimum is retained. There is no arithmetic overflow path.

Invalid dynamic argument conversion signals `CONVERSION_ERROR`.

7.67 intsign

```
intsign(number = .int) = .int
```

Returns `-1`, `0`, or `1` after native signed-integer comparisons. The function does not convert the input to decimal or float.

Invalid dynamic conversion to `.int` signals `CONVERSION_ERROR`.

7.68 lastpos

`lastpos` finds the last match ending at or before a character position:

```
lastpos(needle = .string, haystack = .string, search_end = 0) = .int
```

`search_end` is optional. When omitted, the complete haystack is considered; a supplied value must be a positive 1-based integer. A match is eligible only when its final character is at or before `search_end`. The result is the match's 1-based Unicode character position, or zero for no match or a null needle.

An invalid supplied search end raises `INVALID_ARGUMENTS`.

```
lastpos(" ", "abc def ghi")    /* 8 */
lastpos(" ", "abc def ghi", 7) /* 4 */
lastpos("abc", "abc abc", 6)   /* 1: the later match ends at 7 */
lastpos("aa", "aaa")           /* 2: overlapping matches count */
```

Arguments are read-only. The VM has no reverse substring-search instruction, so the implementation caches both character lengths and performs monotonic `strpos` searches from successive matches. It creates no substrings and permits overlapping candidates.

7.69 left

`left` returns an exact-width left-aligned string:

```
left(string = .string, width = .int, pad = " ") = .string
```

`width` is required and must be non-negative. Longer input is truncated to its leftmost `width` Unicode codepoints; shorter input is padded on the right. `pad` defaults to blank and must contain exactly one codepoint.

Invalid width or pad input raises `INVALID_ARGUMENTS`.

```
left("abc d", 8)      /* "abc d   " */
left("abc d", 8, ".") /* "abc d..." */
left("abc defg", 6)   /* "abc de" */
left("é", 3, "·")     /* "é··" */
```

The source argument is read-only. The implementation caches both character lengths, uses one direct cursor/substring operation for truncation, and one VM `padstr` plus bounded appends for extension. It makes no Level B helper call.

7.70 length

`length` returns the number of Unicode codepoints in a string:

```
length(string = .string) = .int
```

The result is zero for an empty string. It counts characters/codepoints, not UTF-8 storage bytes; a combining codepoint is counted separately from the base character it follows.

```
length("")         /* 0 */
length("abcdefgh") /* 8 */
length("□□é")    /* 3 */
length("é")       /* 2: e plus a combining acute accent */
```

The `.string` argument is read-only and valid in the configured string model, so the Level B surface has no data-error branch. The implementation is one direct VM `strlen` instruction with no helper call or intermediate string.

7.71 linesize

`linesize` returns the implementation's maximum supported line size:

```
linesize() = .int
```

The portable implementation returns `999999999`, the traditional effectively unlimited value used for compatibility with REXX on z/VM. A platform-specific Level B library may replace the module if the host has a smaller native limit.

```
if length(record) > linesize() then say "record is too long"
```

The function has no arguments, allocation, helper call, error path, or Level C BIF counterpart. `.Rexx.linesize()` forwards directly to the same Level B function. `lib/rxfnsb/tests_functional/tlinesz.crexx` checks the exact stable value and repeated calls.

7.72 loadmodule

`loadmodule` is the explicit Level B runtime loader for one RXBIN or RXPLUGIN artifact.

```
module_number = loadmodule(module_path)
```

`module_path` is a `.string`. The return value is an `.int`: a positive value is the one-based number of the last loaded module, while zero or a negative value is the VM loader's failure status.

On success, the VM immediately relinks the runtime so unresolved imports can bind to procedures, classes, and interfaces in the new module. The function loads only the explicit path supplied by the caller; it does not search or sweep a directory. Callers should therefore pass an intended trusted artifact, as described in `docs/books/crexx_programming_guide/running.md`.

7.72.1 Failure contract

`loadmodule` deliberately preserves the `METALOADMODULE` status protocol. A missing, invalid, or blank path returns a non-positive integer rather than raising a signal. This is an explicit VM contract, not silent error handling: the opcode documentation requires callers to inspect the result, and only a fatal dispatch-table allocation failure terminates execution.

```
provider = loadmodule("./providers/example.rxbin")
if provider <= 0 then return 1
```

7.72.2 Coverage and performance

`lib/rxfnsb/tests_functional/ts_loadmodule.crexx` loads a focused provider and checks missing and blank-path statuses in optimized and unoptimized modes. Existing dynamic interface and host regressions cover late binding to real provider exports.

The wrapper is constant-size: one typed string argument, one integer result, one `metaloadmodule` instruction, and no wrapper-side scan, copy, conversion, or allocation. Module loading and relinking costs belong to the VM operation itself and occur only on an explicit call.

7.73 lower

`lower` returns a lowercased copy of a string:

```
lower(string = .string) = .string
```

Every character covered by the VM's deliberately limited, locale-independent simple lowercase table is converted. Other characters and an empty string are unchanged; this is not full Unicode case folding.

```
lower("MiXeD 123 !?") /* "mixed 123 !?" */
lower("ÄÖÜÉ")         /* "äöüé" */
lower("")             /* "" */
```

U+0000 is an ordinary codepoint for this bounded operation; it does not stop conversion of later text.

The argument is read-only and valid `.string` input has no error branch. The implementation is one direct `strlower` instruction and creates only the result string.

This Level B helper lowercases the complete string. It does not accept start or length arguments. `LOWER` is not a required Level C BIF in the repository's Level C catalog; the common runtime's existing helper remains compatibility surface only.

7.74 max

The native Level B `max` function returns the greatest of one or more decimal values:

```
max(first = .decimal, ... = .decimal) = .decimal
```

```
max(100, 99, 1)                             /* 100 */
max(-4, -1, -9)                             /* -1 */
max("9007199254740992", "9007199254740993") /* 9007199254740993 */
```

At least one argument is required by the typed signature. The call boundary performs ordinary `.decimal` conversion for every argument. The implementation uses one linear comparison pass, retains the first value on a tie, allocates no string, and calls no helper.

Invalid dynamic decimal conversion raises the catchable `CONVERSION_ERROR` signal. The separate Level C `MAX` BIF validates Classic `rNUM...` text and preserves the first selected argument's normalized text representation.

7.75 min

The native Level B `min` function returns the least of one or more decimal values:

```
min(first = .decimal, ... = .decimal) = .decimal
```

```
min(100, 99, 1)                              /* 1 */
min(-4, -1, -9)                              /* -9 */
min("9007199254740993", "9007199254740992") /* 9007199254740992 */
```

At least one argument is required by the typed signature. The call boundary performs ordinary `.decimal` conversion for every argument. The implementation uses one linear comparison pass, retains the first value on a tie, allocates no string, and calls no helper.

Invalid dynamic decimal conversion raises the catchable `CONVERSION_ERROR` signal. The separate Level C `MIN` BIF validates Classic `rNUM...` text and preserves the first selected argument's normalized text representation.

7.76 numeric-context accessors

The `numeric` selector exports five zero-argument Level B functions:

```
digits()   = .int
fuzz()     = .int
form()     = .string
numcase()  = .string
standard() = .string
```

Each accessor reads the corresponding setting inherited from its immediate caller. `digits()` returns the positive significant-digit count and `fuzz()` returns the non-negative ignored-digit count. The string accessors return:

Function	Values
`form()`	`scientific`, `engineering`, `unknown`
`numcase()`	`lower`, `upper`, `unknown`
`standard()`	`common`, `classic`, `unknown`

```
example: procedure
  numeric digits 12
  numeric fuzz 2
  numeric form engineering
```

```
numeric case upper
numeric standard classic
say digits() standard() form() fuzz() numcase()
/* 12 classic engineering 2 upper */
```

All five functions are constant-time native context reads. Integer accessors allocate no string; the other accessors only select a constant result name. They have no arguments and no value-dependent error branch.

`numcase()` and `standard()` are CREXX Level B extensions. The repository's Level C catalog requires only the distinct `DIGITS()`, `FORM()`, and `FUZZ()` BIFs; their RexxValue contract is documented in `lib/rxfnsc/numeric.md`.

7.77 objectarrayappend

```
objectarrayappend(array = .object[] expose,
                  value = .object,
                  count = .int optional) = .int
```

`objectarrayappend` copies `value` to the end of `array` `count` times and returns the new high-water mark. `count` defaults to one. Ordinary Level B object assignment has value-copy semantics; appended slots are not shared references to the caller's object.

A zero count is a no-op. A negative count raises `INVALID_ARGUMENTS`.

The exposed object array is mutated in place. The implementation uses one VM `insattrs1` bulk growth operation followed by one object-value assignment per new slot; existing values are not moved in REXX code. The focused harness is `lib/rxfnsb/tests_functional/ts_objectarrayappend.crexx`.

7.78 objectarraydelete

```
objectarraydelete(array = .object[] expose,
                  from = .int,
                  count = .int) = .int
```

`objectarraydelete` removes up to `count` object values beginning at the positive one-based index `from`, shifts the remaining tail down, and returns the new high-water mark.

A zero count, empty array, or start beyond the high-water mark is a no-op. An overlong count clamps to the available tail without overflowing native index arithmetic. A position below one or negative count raises `INVALID_ARGUMENTS`.

The exposed array is mutated in place through one VM `delattrs1` bulk pointer shift; object values in the retained tail are not moved by a Rexx-level loop. The focused harness is `lib/rxfnsb/tests_functional/ts_objectarraydelete.crexx`.

7.79 objectarraydrop

```
objectarraydrop(array = .object[] expose) = .int
```

`objectarraydrop` clears every logical element from an object array in place and returns zero, the new high-water mark. Clearing an empty array is an idempotent no-op, and the same array can be populated again afterward.

The implementation is one VM `setattrs array,0` operation. It does not walk or copy object values in REXX code. The focused harness is `lib/rxfnsb/tests_functional/ts_objectarraydrop.crexx`.

7.80 objectarrayinsert

```
objectarrayinsert(array = .object[] expose,
                  from = .int,
                  count = .int,
                  value = .object) = .int
```

`objectarrayinsert` opens `count` slots at the positive one-based position `from`, shifts the existing tail upward, copies the `value` object into every new slot, and returns the new high-water mark. Ordinary Level B object assignment has value-copy semantics; this helper does not create shared references.

A position beyond the end clamps to append. A zero count is a no-op. A position below one or negative count raises `INVALID_ARGUMENTS`.

The exposed object array is mutated in place. The implementation uses one VM `insattrs1` bulk pointer shift followed by one object-value assignment per new slot; it does not move the existing tail in REXX code. The focused harness is `lib/rxfnsb/tests_functional/ts_objectarrayinsert.crexx`.

7.81 objectarraymove

```
objectarraymove(array = .object[] expose,
```

```
                 from = .int,
                 count = .int,
                 to = .int) = .int
```

`objectarraymove` moves up to `count` object values beginning at the positive one-based `from` position so they are inserted at `to`, interpreted in the original array. The order of moved and retained values is preserved, and the unchanged high-water mark is returned.

An overlong count clamps to the source tail. A destination beyond the end clamps to append. A zero count, empty array, source beyond the end, or destination within/immediately after the selected block is a no-op. A source or destination below one or negative count raises `INVALID_ARGUMENTS`.

The implementation inserts the destination gap first, copies each object value directly once into a non-overlapping target, then removes the original block with one `delattrs1`. It avoids the former temporary object array and second deep-copy pass. The focused harness is `lib/rxfnsb/tests_functional/ts_objectarraymove.crexx`.

7.82 objectarrayprepend

```
objectarrayprepend(array = .object[] expose,
                   value = .object,
                   count = .int optional) = .int
```

`objectarrayprepend` copies `value` to the front of `array` `count` times and returns the new high-water mark. `count` defaults to one. Ordinary Level B object assignment has value-copy semantics; prepended slots are not shared references to the caller's object.

A zero count is a no-op. A negative count raises `INVALID_ARGUMENTS`.

The exposed object array is mutated in place. The implementation uses one VM `insattrs1` bulk shift followed by one object-value assignment per new slot; existing values are not moved in REXX code. The focused harness is `lib/rxfnsb/tests_functional/ts_objectarrayprepend.crexx`.

7.83 overlay

`overlay` writes formatted text over a target string:

```
overlay(new_string = .string, target = .string, start = 1,
        overlay_length = 0, pad = " ") = .string
```

`start` is an optional positive 1-based character position. When `overlay_length` is omitted it is the character length of `new_string`; an explicitly supplied zero writes no new text. The formatted overlay is truncated or padded to that width. If `start` is beyond the target, the target is padded up to the preceding character. `pad` defaults to blank and must contain exactly one Unicode codepoint.

Invalid start, length, or pad input raises `INVALID_ARGUMENTS`.

```
overlay("qq", "abcd")               /* "qqcd" */
overlay(".", "abcdef", 3, 2)        /* "ab. ef" */
overlay("123", "abc", 5, 6, "+") /* "abc+123+++" */
overlay("foo", "abcdef", 3, 0)   /* "abcdef" */
```

Arguments are read-only. The implementation caches character lengths and uses direct VM substring, append, and pad operations. It does not call `LEFT`, `SUBSTR`, `COPIES`, or another Level B formatting helper.

7.84 parse

`parse` is the convenience wrapper around `parsecompile` and `parsestring`:

```
parse(source = .string,
      template = .string,
      expose variable = .string[],
      expose variable_content = .string[],
      strip_option = .int optional,
      case_option = .int optional) = .int
```

It returns the number of variables and fills aligned name and value arrays. `strip_option` is `0` to preserve values or `1` to remove leading and trailing whitespace. `case_option` is `0` to preserve case, `1` for uppercase, or `2` for lowercase. Transformations are applied after parsing, with stripping first.

```
names = .string[]
values = .string[]
count = parse("  ALPHA , BETA  ", "first','second", names, values, 1,
    2)
/* count = 2; values[1] = "alpha"; values[2] = "beta" */
```

Invalid option values or an invalid dynamic template signal `INVALID_ARGUMENTS`. Template validation occurs before the exposed output arrays are changed. See parsecompile for the complete legacy template format.

The wrapper uses direct VM strip and case operations. It is a Level B runtime helper, not a Level C BIF.

7.85 parsecompile

parsecompile validates and compiles the legacy dynamic-template format used by parse and parsestring:

```
parsecompile(template = .string,
             expose token = .string[],
             expose token_type = .int[]) = .int
```

The return value is the number of plan items. token contains their payloads and token_type contains integer kinds:

Kind	Meaning	Example payload
1	variable	name
2	quoted literal, without its outer quotes	,
3	absolute position	10
4	signed relative position	+8
5	implicit word separator	the original whitespace run
6	prior-variable delimiter search	name from (name)

For example, first','second third produces variable first, literal ,, variable second, an implicit separator, and variable third. Whitespace is preserved as kind 5 only between adjacent variables; whitespace touching a quoted literal is suppressed. Single- and double-quoted literals are accepted, and doubled quote pairs remain doubled in the literal payload.

The recognised whitespace characters are space, tab, CR, LF, vertical tab, form feed, and U+00A0. Positions must fit a signed 64-bit native integer.

Malformed quotes, signed positions, parenthesized searches, or oversized positions signal INVALID_ARGUMENTS. Validation finishes before either exposed array is cleared, so both arrays remain unchanged when a signal is raised.

This is a Level B runtime helper, not a Level C BIF. The compiler PARSE exit does not use this legacy dynamic-template representation: it validates the source template in the exit and emits direct instructions or a packed parseplan descriptor.

7.86 parsestring

`parsestring` executes the legacy parallel-array plan produced by `parsecompile`:

```
parsestring(parse_string = .string,
            tokenhi = .int,
            expose token = .string[],
            expose token_type = .int[],
            expose variable = .string[],
            expose variable_content = .string[],
            template = "unknown")
```

`token` and `token_type` are borrowed read-only references. The output arrays are cleared and populated in template-variable order, with names in `variable` and matching fields in `variable_content`. Absolute and relative positions use one-based source positions; literal and prior-variable searches begin at the current parse position. Kind 5 treats space, tab, CR, LF, vertical tab, form feed, and U+00A0 as whitespace.

For example, compiling and executing `left','middle','right` against `one,two,three` yields `one`, `two`, and `three`. A search item such as `(left)` uses the value already assigned to `left` as the next delimiter.

The function checks `tokenhi`, every integer kind, and all position, separator, and search payloads before clearing the output arrays. A malformed or truncated plan signals `INVALID_ARGUMENTS` and leaves existing outputs unchanged. The source and both plan arrays are never changed.

The implementation scans the source directly with VM string operations and does not copy the unconsumed source tail for each delimiter search. This is a Level B runtime helper, not a Level C BIF.

7.87 pos

`pos` finds the first occurrence of text at or after a character position:

```
pos(needle = .string, haystack = .string, start = 1) = .int
```

`start` is an optional positive 1-based integer. The result is the 1-based Unicode character position of the match, or zero when no match exists. A null needle or haystack also returns zero.

An invalid start raises `INVALID_ARGUMENTS`.

```
pos("day", "Saturday")      /* 6 */
pos(" ", "abc def ghi", 5) /* 8 */
pos("→", "→→abc", 3)       /* 4 */
pos("", "anything")        /* 0 */
```

Arguments are read-only. The implementation performs direct null checks and one VM `strpos` search. It makes no LENGTH or other Level B helper call and allocates no substring.

The VM search is length bounded: U+0000 can occur in either argument and does not terminate matching.

7.88 qextractall

`qextractall(open=.string, close=.string, text=.string [,start=.int [,mode=.string]]) = .string[]` returns every balanced top-level span in source order. `X/E` returns contents and `I/C` includes delimiters. It scans once and selects spans by position. See the shared quote-aware contract.

7.89 qextractpair

`qextractpair(open=.string, close=.string, text=.string [,start=.int [,mode=.string]]) = .string` returns the first balanced top-level span. `X/E` returns its contents; `I/C` includes delimiters. No opening pair returns an empty string. Invalid mode or grammar signals `INVALID_ARGUMENTS`. See the shared quote-aware contract.

7.90 qpos

`qpos(needle=.string, text=.string [,start=.int]) = .int` returns the first one-based codepoint position of a non-empty `needle` outside quotes, or `0`. `start` must be positive. Invalid arguments and unmatched quotes signal `INVALID_ARGUMENTS`. See the shared quote-aware contract.

7.91 qremoveall

`qremoveall(open=.string, close=.string, text=.string [,mode=.string]) = .string` removes every balanced top-level span. `I/C` (the default) removes delimiters and contents; `X/E` removes contents only. Positional reconstruction handles repeated equal spans correctly. See the shared quote-aware contract.

7.92 qsplit

`qsplit(text=.string, separator=.string) = .string[]` splits only outside quotes. The separator must be non-empty. Results preserve source whitespace, quote delimiters, adjacent empty fields, and a trailing empty field. See the shared quote-aware contract.

7.93 qsplitsafe

`qsplitsafe(text=.string, separator=.string [,start=.int [,pairs=.string]]) = .string[]` additionally suppresses splits inside nested one-codepoint delimiter pairs. `pairs` is an opener/closer sequence such as `()[]`; the default is `()`. The prefix before `start` stays in the first field. Invalid pair grammar signals `INVALID_ARGUMENTS`. See the shared quote-aware contract.

7.94 qstripcomment

`qstripcomment(open=.string [,close=.string], text=.string) = .string` removes comments outside quotes. Omitted or empty `close` selects line comments and preserves CRLF, LF, and CR line endings exactly. A non-empty `close` selects nested balanced block comments. See the shared quote-aware contract.

7.95 qsubword

`qsubword(text=.string, word_number=.int [,count=.int]) = .string` returns a source span beginning at the positive word number. Omitted `count` selects through the last word; explicit zero returns empty; a negative count signals `INVALID_ARGUMENTS`. Original separators between selected words are preserved. See the shared quote-aware contract.

7.96 quote-aware text contract

The `q*` string functions share one positional Unicode scanner. They are Level B library functions, not classic Level C BIFs.

Single (`'`) and double (`"`) ASCII quotes can begin anywhere in a word. A matching doubled quote represents an escaped quote. Quote delimiters remain part of returned words and fields; an unmatched quote signals `INVALID_ARGUMENTS`. Positions and lengths are Unicode codepoints. Word separation uses the Unicode 17.0 `White_Space` property, not grapheme boundaries.

Balanced-pair functions ignore delimiters inside quotes and support nested pairs. Empty, identical, mismatched, or unclosed delimiters signal `INVALID_ARGUMENTS`. Split and removal functions reconstruct results from source spans, so repeated equal values, empty fields, whitespace, and line endings are preserved exactly.

`ts_qpos.crexx`, `ts_qwordlength.crexx`, and `ts_qlibrary.crexx` run in optimized and unoptimized Level B modes and cover the shared grammar, Unicode positions, exact reconstruction, and signal paths.

7.97 qword

`qword(text=.string, word_number=.int) = .string` returns a positive one-based quote-aware word, including its quote delimiters, or empty when absent. Quotes may be attached to unquoted text. See the shared quote-aware contract.

7.98 qwordindex

`qwordindex(text=.string, word_number=.int) = .int` returns the one-based codepoint start of a positive quote-aware word, or `0` when absent. See the shared quote-aware contract.

7.99 qwordlength

`qwordlength(text=.string, word_number=.int) = .int` returns the codepoint length of a positive one-based quote-aware word, including its quote delimiters. An absent word returns `0`; an invalid word number or unmatched quote signals `INVALID_ARGUMENTS`.

The implementation reads the selected span length from the shared scanner and does not copy the word. There is no Level C BIF or class method named QWORDLENGTH. See the shared quote-aware contract.

`ts_qwordlength.crexx` covers quoted and unquoted lengths, Unicode whitespace, missing input, non-mutation, and signals in both Level B build modes.

7.100 qwordpos

`qwordpos(search=.string, text=.string [,start=.int]) = .int` finds an exact sequence of quote-aware words at or after the positive word number `start`. Partial-word matches do not count. It returns `0` for an empty or absent phrase. See the shared quote-aware contract.

7.101 qwords

`qwords(text=.string) = .int` counts quote-aware words in one scan. Unicode 17.0 `White_Space` separates words. See the shared quote-aware contract.

7.102 raise

`raise` is the internal `_rxsysb` helper used by Level B runtime code to raise one of the VM's canonical signals with a diagnostic message.

```
call raise "SYNTAX", "Error 40.28: invalid DATE argument 3", format
```

All three arguments are `.string`. `type` normally names a canonical VM signal such as `INVALID_ARGUMENTS` or `NOTREADY`. The compatibility spellings `syntax`, `SYNTAX`, and `error` are normalized to the VM's canonical `ERROR` signal. `code` is the primary diagnostic text and `parm1` supplies contextual text. When `parm1` is not blank, the delivered signal message is `code`, one blank, then `parm1`; a blank `parm1` delivers `code` unchanged.

The procedure retains its internal compatibility result `.int`. Normally the signal transfers control and no result is observed. A signal handler may explicitly skip the signal, in which case the helper returns zero to its caller. An unknown `type` is rejected by the VM as `INVALID_SIGNAL_CODE`.

7.103 Coverage and performance

`lib/rxfnsb/tests_functional/ts_raise.crexx` verifies the delivered signal name and message, the blank-context case, and VM validation of an unknown signal name in optimized and unoptimized selector overlays.

The helper performs no output and no scan. It builds at most one diagnostic string and executes one dynamic-name `signal` instruction.

7.104 random

The typed Level B function returns a pseudo-random `.int`:

```
random([minimum = .int [, maximum = .int [, seed = .int]]]) = .int
```

With no arguments the range is `0..999`. One argument is the inclusive maximum and therefore selects `0..maximum`. In the positioned three-argument form an omitted minimum defaults to `0`, an omitted maximum defaults to `999`, and an optional seed resets the module-scoped sequence before the value is drawn.

Arguments must be non-negative, the minimum must not exceed the maximum, and their difference must not exceed `100000`. Violations signal `INVALID_ARGUMENTS`.

The generator is deterministic Park-Miller state. Seed zero is normalized to seed one; omitted seeding continues the current module sequence. Bounded values use rejection sampling rather than `%` over a C-library `rand()` result, so the range is unbiased. This generator is suitable for language-level repeatable sequences, not cryptography.

The implementation has no VM-global RNG dependency. `RexxRandomState` is also the state object used by the separately scoped Level C configuration service.

7.105 reradix

`reradix` converts an unsigned integer between radices 2 through 16:

```
reradix(subject=.string, FromRadix=.int, ToRadix=.int) = .string
```

The source uses ASCII digits `0` through `9` and letters `A` through `F` case-insensitively. The result uses uppercase digits and has no numeric-size limit imposed by the native `.int` type.

```
reradix("F81", 16, 10) == "3969"
reradix("340282366920938463463374607431768211455", 10, 16) == ,
  "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"
```

Leading zeros are normally suppressed. Binary-to-hex conversion preserves one hex digit for each supplied group of up to four bits, and hex-to-binary preserves four bits per supplied hex digit:

```
reradix("00001", 2, 16) == "01"
reradix("0F", 16, 2) == "00001111"
```

An empty subject, a radix outside 2 through 16, a non-ASCII digit, or a digit outside the source radix signals `INVALID_ARGUMENTS`. The source is not modified.

The specialized binary/hex paths are linear. Other pairs use little-endian target digits and bounded multiply/add intermediates, avoiding native-integer overflow, decimal/scientific conversion, front-prepending, and helper calls. `.Rexx.reradix(fromRadix, toRadix)` forwards to this Level B function. There is no Level C BIF named RERADIX.

7.106 reverse

`reverse` returns the Unicode codepoints of a string in reverse order:

```
reverse(string = .string) = .string
```

The function reverses codepoints, not UTF-8 bytes. It does not combine or reorder grapheme clusters: for example, a combining mark is an independent codepoint. Empty and one-codepoint strings are returned unchanged, and the source is read-only. Every valid `.string` is accepted, so there is no domain error branch.

```
reverse("abc")     /* "cba" */
reverse("□□aé") /* "□□éa" */
reverse("")        /* "" */
```

The implementation caches the character length and makes one reverse pass with the VM `strchar` and `appendchar` operations. It makes no Level B helper call and does not repeatedly copy the growing result with string concatenation.

7.107 right

`right` returns an exact-width right-aligned string:

```
right(string = .string, width = .int, pad = " ") = .string
```

`width` is required and must be non-negative. Longer input is truncated to its rightmost `width` Unicode codepoints; shorter input is padded on the left. `pad` defaults to blank and must contain exactly one codepoint.

Invalid width or pad input raises `INVALID_ARGUMENTS`.

```
right("abc  d", 8)  /* "  abc  d" */
right("abc def", 5) /* "c def" */
right("12", 5, "0") /* "00012" */
right("é", 3, "·")  /* "··é" */
```

The source argument is read-only. The implementation caches both character lengths, uses one direct cursor/substring operation for truncation, and one VM `padstr` plus bounded appends for extension. It makes no Level B helper call.

7.108 sequence

`sequence` builds an inclusive ascending range of Unicode characters:

```
sequence(from=.string, tos=.string) = .string
```

Both endpoints must contain exactly one Unicode character. The result includes both endpoints and every Unicode scalar value between them. Surrogate code points U+D800 through U+DFFF are not Unicode scalar values and are skipped.

```
sequence("a", "f") == "abcdef"
sequence("□", "□") == "□□□"
```

A descending range or an empty/multiple-character endpoint signals `INVALID_ARGUMENTS`; SEQUENCE never wraps. Use the distinct Level B `xrange` helper for the legacy wrapping U+0000 through U+00FF domain. Inputs are not modified.

The implementation reads each endpoint code point once and appends the result in one linear pass with no Level B helper call or intermediate range array. `.Rexx.sequence(final)` forwards its receiver and `final` to this function. There is no Level C BIF named SEQUENCE.

`lib/rxfnsb/tests_functional/ts_sequence.crexx` covers ASCII, Greek and supplementary characters, the surrogate gap, the highest scalar value, non-mutation, and every endpoint error family.

7.109 sign

The native Level B function returns the sign of a decimal value:

```
sign(number = .decimal) = .int
```

It returns `1` for a positive value, `-1` for a negative value, and `0` for either signed representation of zero. The decimal boundary preserves signs for values outside binary floating-point range:

```
sign("1E-999")  /*  1 */
sign("-1E+999") /* -1 */
sign("-0")      /*  0 */
```

The implementation performs at most two decimal comparisons and allocates nothing. It does not format or modify the argument.

The Level B call boundary performs ordinary `.decimal` conversion. Invalid dynamic conversion raises the catchable `CONVERSION_ERROR` signal.

This is the strongly typed foundation API. The distinct Level C `SIGN` BIF normalizes Classic numeric text and reports standard context errors.

7.110 Level B signal objects

`signal.crexx` defines the typed Level B condition objects used by signal handlers and the action values returned by action-aware handlers. It is runtime support for Level B; it is not a Level C Classic BIF.

7.111 Public condition contract

`.signal(name, message="")` creates a message-only `.standard_signal`. Names are normalized to uppercase. `code()` maps every VM signal name to its integer runtime code; `SYNTAX` is an alias for `ERROR` (code 3), and an unknown name maps to `INVALID_SIGNAL_CODE` (code 13). Raising an unknown dynamic name causes the VM to deliver `INVALID_SIGNAL_CODE`. The concurrency runtime uses `CHANNEL_ERROR` (code 26) for catchable channel/class lifecycle failures and `TASK_FAILURE` (code 27) for typed task-result demand failures.

The `.signal` methods are:

Method	Type	Meaning
`name()`	`.string`	Canonical condition name
`code()`	`.int`	VM signal code
`module()`	`.int`	One-based originating module, or zero when absent
`address()`	`.int`	Originating RXAS address, or zero when absent
`message()`	`.string`	String message view
`payload()`	`.object`	Object view of the payload slot
`file()`	`.string`	Closest preceding source file, or blank
`line()`	`.int`	Source line, or zero
`column()`	`.int`	Active source column, or zero
`source()`	`.string`	Whole authored source line, or blank

Message-only `.standard_signal` values have no runtime origin, so their module/address/line/column methods return zero and file/source return blank.

7.112 Runtime transport

The VM passes a five-slot interrupt object to a procedure handler: code, module, address, name, and message/payload. `.runtime_signal_raw` maps those physical slots with explicit register views. `.runtime_signal` wraps the raw object behind the public `.signal` interface and resolves source metadata only when the file/line/column/source methods are requested.

The raw class and `runtime_signal.set_raw()` are runtime/compiler transport, not ordinary application construction APIs. Their attribute order and register views must remain aligned with `rxsignal_populate_raw_interrupt()` in the VM.

7.113 Handler actions

Action-aware handlers return one of:

```
return .signalaction.skip()
return .signalaction.fail()
```

`kind()` returns `skip` or `fail`. Skip resumes after the faulting instruction and fail propagates the condition. Instruction-level retry is deliberately not provided; use an explicit source-level loop for retryable work.

7.114 Coverage and performance

`lib/rxfnsb/tests_functional/ts_signal.crexx` covers the public factory provider, the complete name/code table, all action factories, raw transport views, the runtime wrapper, populated source metadata in the unoptimized image, and the documented blank/zero metadata results in the source-stripped optimized image. Compiler-exit and VM RXAS tests separately cover syntax lowering, handler diagnostics, scope, and dispatch modes.

The name/code mapping uses a static string SELECT so optimized RXAS can use one packed dispatch table instead of repeating a long comparison ladder. Ordinary field access is constant-time. Source metadata lookup is a closest-preceding linear scan and runs only when a handler requests source information.

7.115 space

`space` normalizes whitespace-delimited words:

```
space(string = .string, count = 1, pad = " ") = .string
```

`count` is an optional `.int` and must be non-negative. `pad` defaults to blank and must contain exactly one Unicode codepoint. Leading and trailing Unicode whitespace is removed, internal whitespace runs delimit words, and the words are joined with exactly `count` copies of `pad`. A zero count joins the words without a separator. Invalid count or pad input raises `INVALID_ARGUMENTS`.

```
space("abc   def   ")          /* "abc def" */
space("  abc def ", 3)         /* "abc   def" */
space("abc   def   ", 0)       /* "abcdef" */
space(" ab", 1, "·")          /* "a·b" */
```

The source is read-only. The implementation scans it once with the VM Unicode blank/nonblank instructions and appends each word slice directly. It constructs the separator once with `padstr`, lazily when a second word needs it; there are no Level B `WORDS`, `WORD`, or `COPIES` calls and no repeated result concatenation.

7.116 splice

```
splice(replacement = .string,
       source = .string,
       at = .int,
       remove_length = .int) = .string
```

`splice` removes up to `remove_length` characters from `source` beginning at the positive one-based position `at`, then inserts `replacement` at that position. It does not pad. The result may grow, shrink, or keep the same length.

```
splice("X", "abcdef", 3, 2)     /* "abXef" */
splice("-", "abcdef", 4, 0)     /* "abc-def" */
splice("", "abcdef", 4, 99)     /* "abc" */
```

A position beyond `length(source) + 1` clamps to append. An overlong removal clamps to the available tail. A position below one or a negative removal length raises `INVALID_ARGUMENTS`.

All offsets count Unicode characters rather than UTF-8 bytes. The inputs are not mutated. The implementation uses direct bounded cursor slices and does not call `length`, `left`, `substr`, or another selector. The focused harness is `lib/rxfnsb/tests_functional/ts_splice.crexx`.

7.117 .stem class

This document defines the stable Level B `.stem` and `.stemIterator` contract.

7.117.1 Overview

A `stem` is a string-to-string dictionary supporting CREXX property and bracket notation for Classic-style compound-variable tails. A missing key returns the assigned stem default, or the empty string before a default is assigned. Use `size()` or the key arrays when absence must be distinguished from an assigned empty value.

7.117.2 API

- `get(key = .string) = .string` returns the tail value, the stem default, or the empty string.
- `set(key = .string, value = .string) = .void` inserts or replaces a value. Property assignment with an omitted key, `stem. = value`, assigns the Classic stem default instead; an explicitly supplied empty key is still a normal key. Assigning a default also replaces all values belonging to existing tails.
- `size() = .int` returns the number of distinct keys.
- `key(index = .int) = .string`, `value(index = .int) = .string`, and `valueAt(index = .int) = .string` access the internal one-based insertion arrays. An index outside `1..size()` signals `INVALID_ARGUMENTS`.
- `tails() = .string[]` and `values() = .string[]` return independent arrays.
- `iterator() = .stemIterator` creates a live iterator; `snapshotIterator() = .stemIterator` creates a snapshot.

The iterator factory accepts only snapshot flags `0` and `1`; another flag signals `INVALID_ARGUMENTS`.

7.117.3 Implementation (hash map with separate chaining)

The implementation uses parallel arrays to manage a 256-bucket hash map:

- `buckets`: An array of 256 integers pointing to the head of a chain.
- `keys`: A flat array storing string keys.
- `vals`: A flat array storing string values.
- `value_generations`: the default generation in which each explicit value was assigned.
- `next`: An integer array storing the index of the next item in the chain.

Stem-default assignment is constant time. It stores the new default and advances the default generation; older explicit values then read as the default without rewriting the value array. A later explicit tail assignment records the current generation and becomes visible again. Indexed access, extracted values, and iterators all resolve the effective value through the same generation rule.

The hash loop reads Unicode codepoints directly with VM string instructions and uses a multiplier of 31. Each step is reduced modulo 256, which is equivalent for the final bucket and prevents native integer overflow for long keys.

7.117.4 Usage

Stems can be accessed using standard `get` and `set` methods, but the preferred approach is using CREXX object property syntax (`obj.key` or `obj["key"]`) which is automatically rewritten to method calls by the compiler.

```
s = .stem()

s. = "fallback"
say s.unknown              /* fallback */

/* Property syntax (Syntactic Sugar) */
s.key = "value"
val = s.key

/* Multi-tail property syntax preserves separators in the key */
customer = "acme"
invoice = "2026.05"
s.customer.invoice = "value4"  /* key is "acme.2026.05" */

/* Bracket notation for keys that aren't valid identifiers */
s["my-key"] = "value2"

/* Direct method calls */
call s.set("other_key", "value3")
val = s.get("other_key")
```

7.117.5 Iteration

Stems are keyed containers, so iteration is over tails/keys rather than list positions. Iteration order is unspecified. The current Rexx implementation stores keys in insertion order, but callers must not depend on that ordering.

```
s = .stem()
s.name = "Ada"
s.lang = "Rexx"

it = s.iterator()
loop while it.hasNext()
  tail = it.next()
  say tail "=" it.value()
end
```

`iterator()` returns an unsynchronized live iterator. It observes the current stem through a weak reference; value updates after iterator creation are visible, and newly added tails may be observed by the current implementation.

`snapshotIterator()` copies the stem's tails and values once in the iterator factory. Use it when callers need stable factory-time contents while the stem may be mutated.

The iterator API is:

- `hasNext() = .int`
- `next() = .string`, returning the next tail/key
- `value() = .string`, returning the value for the current tail
- `index() = .int`
- `reset() = .void`
- `isLive() = .int`

7.117.6 Performance

Hashing uses direct `strlen` and `strchar` VM operations and makes no Level B selector calls. Lookup and update use separate bucket chains with early exit. The implementation remains Level B so it can evolve with the standard typed array surface; no native-C replacement is required by this contract.

7.117.7 Notes on Character Encoding

VM string parsing exposes Unicode codepoints rather than UTF-8 bytes, so equal Unicode keys hash consistently regardless of their encoded byte width.

7.118 strip

`strip` removes leading, trailing, or both runs of unwanted codepoints:

```
strip(string = .string, option = "B", char = " ") = .string
```

Only the first codepoint of `option` is significant, case-insensitively: `L` removes a leading run, `T` removes a trailing run, and `B` removes both. An invalid or empty option raises `INVALID_ARGUMENTS`.

When `char` is omitted, the configured Unicode whitespace set is removed. When it is supplied, it must contain exactly one codepoint and only that codepoint is removed; consequently, an explicitly supplied blank differs from omission for other Unicode whitespace. An invalid supplied char raises `INVALID_ARGUMENTS`.

```
strip("  ab c  ")          /* "ab c" */
strip("  ab c  ", "L")     /* "ab c  " */
strip("12.7000",, "0")     /* "12.7" */
strip(" ab  c")          /* "ab c" */
strip("  x",, " ")       /* " x" */
```

The source is read-only. The implementation calculates one start/end slice, using direct VM Unicode blank scans when `char` is omitted and direct codepoint comparisons when it is supplied. It performs at most one result substring and makes no Level B helper call.

7.119 substr

`substr` returns a 1-based Unicode-codepoint slice:

```
substr(string = .string, start = .int, length = 0, pad = " ") = .string
```

`start` is required and must be positive. When `length` is omitted, the result runs through the end of the source, or is empty when `start` is beyond it. A supplied `length` is an `.int` and must be non-negative; the result has exactly that many codepoints and is padded on the right when the source is short. `pad` defaults to blank and must contain exactly one codepoint. Invalid start, supplied length, or pad input raises `INVALID_ARGUMENTS`.

```
substr("abc", 2)          /* "bc" */
substr("abc", 2, 4)       /* "bc  " */
substr("abc", 5, 4)       /* "    " */
substr("abc", 2, 6, ".")  /* "bc...." */
substr("□□René", 5, 2)  /* "□□" */
```

The source is read-only. The implementation caches its character length, takes at most one direct VM substring, and appends padding directly with `padstr` only when needed. It makes no Level B helper call and does not use the mutating `substcut` instruction.

7.120 substro

`substro` is a cREXX-specific alternate name with the same Level B slicing contract as `substr`:

```
substro(string = .string, start = .int, length = 0, pad = " ") = .
    string
```

`start` is required and must be positive. When `length` is omitted, the result runs through the end of the source, or is empty when `start` is beyond it. A supplied `length` is an `.int` and must be non-negative; the result has exactly that many codepoints and is padded on the right when the source is short. `pad` defaults to blank and must contain exactly one codepoint. Invalid start, supplied length, or pad input raises `INVALID_ARGUMENTS`.

```
substro("abc", 2)          /* "bc" */
substro("abc", 2, 4)       /* "bc  " */
substro("abc", 5, 4)       /* "    " */
substro("abc", 2, 6, ".")  /* "bc...." */
```

The source is read-only. The implementation caches its character length, takes at most one direct VM substring, and appends padding directly with `padstr` only when needed. It makes no Level B helper call. `SUBSTRO` is not in the repository Level C BIF catalog and therefore has no Level C implementation.

7.121 subword

```
subword(string = .string, start = .int [, count = .int]) = .string
```

`subword` returns up to `count` words beginning at the positive one-based `start`. Omitted count selects through the final word; explicit zero returns empty; and an oversized count clamps to the remaining words. A start beyond the available words returns empty.

Leading and trailing whitespace is excluded while all whitespace between the selected words is preserved. Words use the VM Unicode whitespace definition. A non-positive start or supplied negative count signals `INVALID_ARGUMENTS`.

The implementation scans the source once to identify one span and extracts it once, without other selector calls. The focused harness is `lib/rxfnsb/tests_functional/ts_subword.crexx`.

7.122 symbol

`symbol(name)` classifies a candidate against the compiled Level B source scope at the call site.

```
count = 3
say symbol("count")  /* VAR */
say symbol("unused") /* LIT */
say symbol("a*b")    /* BAD */
```

The `.string` result is:

- `BAD` when the text is empty, contains a character outside ASCII letters, digits, underscore, and period, or is a Level B reserved word;
- `VAR` when current caller metadata identifies the name as a visible variable or constant binding;
- `LIT` when the name is valid but has no visible binding.

Level B has reserved words and compiled typed locals, so this contract is deliberately different from the Classic Level C `SYMBOL` BIF. Invalid candidate text is normal classification and returns `BAD`; it is not a runtime error and does not raise a signal.

7.123 Coverage and performance

`lib/rxfnsb/tests_functional/ts_symbol.crexx` covers visible integer/string variables, literal values, unused names, exact reserved-word matching, dotted constants, underscore names, empty text, and invalid characters in optimized and unoptimized selector overlays.

The implementation validates the input once, normalizes it once, precomputes the exact keyword and metadata probes, then scans caller metadata backwards only as far as the current procedure boundary.

7.124 time

`time` exposes the Level B VM clocks through a text-returning API:

```
time([option=.string]) = .string
```

The case-insensitive options are:

Option	Result
`N` or omitted	local `hh:mm:ss`
`L`	local `hh:mm:ss.ffffff`
`C`	local 12-hour `hh:mmam` or `hh:mmpm`
`H`, `M`, `S`, `US`	hour, minutes, seconds, or microseconds since local midnight
`E`, `R`	elapsed seconds, or elapsed seconds followed by reset
`UTC`	UTC `hh:mm:ss`
`ZD`	UTC offset in seconds
`T`, `TS`	process clock ticks, or ticks per second
`ZN`	standard and daylight timezone names separated by `;`

Unsupported options signal `INVALID_ARGUMENTS`. Platform clock failures are not reported by the current VM instructions; allocation failure is fatal.

The `US`, offset, tick, and timezone paths return immediately after one clock instruction. Numeric decomposition occurs only for formats that require it; normal, long, and civil text is appended directly without general string helper calls.

The elapsed timer is process-global through `_rxsysb._elapsed`, handles local-midnight rollover, and `R` returns the prior elapsed value before reset.

This typed Level B extension is distinct from Classic Level C `TIME`, whose three-argument conversion and frozen-clause-time contract is documented in `lib/rxfnsc/time.md`.

7.125 Level B trace runtime

`trace.crexx` is the shared Level B runtime used by the certified TRACE exit, RXDB, and ADDRESS command tracing. It is not the Classic `TRACE()` built-in function; that separate Level C contract is documented in `lib/rxfnsc/trace.md`.

7.126 Public objects

7.126.1 `trace_interrupt_raw`

This is the VM breakpoint-signal transport view. Its register-backed fields are `code` (register 1), `module` (2), `address` (3), and `name` (4). Their order and register mappings are ABI, not implementation detail. The four accessor methods return the corresponding typed value.

7.126.2 `tracecontext`

The factory accepts typed `module` and `addr` integers plus optional string `mode`, integer `signal_code`, and string `signal_name`. Accessors expose the signal identity, module/address, mode, source file/line/column/text, formatted source line, closest source, decoded ASM line, and procedure name.

Metadata is cached per context. Direct controller contexts prepare their content before return. Breakpoint contexts first resolve the procedure needed for filtering; rejected events do not scan source or decode an instruction. Accepted REXX contexts load source data, while ASM/LLM contexts also decode the instruction. Repeated accessors do not rescan metadata.

7.126.3 `tracecontroller`

The controller groups these public operations:

- mode and breakpoint control: `mode`, `set_mode`, `toggle_mode`, `enable_breakpoints`, and `disable_breakpoints`;
- module policy: `set_first_client_module`, `first_client_module`, `set_latest_-module_only`, `latest_module_only`, `set_include_runtime`, and `include_-runtime`;
- namespace policy: reset, suppress, unsuppress, add/remove, cache clearing, containment, component matching, and `namespace_is_suppressed`;
- event handling: `context`, `context_from_interrupt`, `should_trace`, and `should_-trace_module`;
- metadata/status queries: exact/closest source, ASM line, procedure name, loaded module count/name, module loading, loaded procedure count/name/id, and procedure search;
- trace-result coordination: clear/capture target, pending result/module/ prefix/type/register/constant, supplied result/value, and parent-value supply.

Module and procedure indices are `.int`. Invalid metadata indices return the documented empty-string or zero status. `load_module` retains the VM's module number/non-positive status protocol.

7.127 Namespace helpers

The 30 exported `_trace_*` helpers are the compiler-exit-facing surface. They cover runtime initialization, mode/environment/format/output selection, namespace controls, event filtering, result-register coordination, escaping, line output, and ADDRESS command-before/after records. Their argument and return metadata is typed; counts, addresses, modules, return codes, masks, and registers are integers.

Supported runtime modes include the Classic `A`, `C`, `E`, `F`, `I`, `L`, `N`, `O`/`OFF`, and `R` families plus CREXX `REXX`, `ASM`, and `LLM`. Full-word left prefixes are normalized once. Signed non-zero settings select Normal and zero selects Off. An invalid `_trace_set` mode raises `INVALID_ARGUMENTS` and leaves the previous state intact.

Output targets are stdout, stderr, or an append-mode file. An OS open failure raises NOTREADY; it is not printed and ignored. Environment-driven invalid settings deliberately retain their documented policy: turn tracing off and emit a trace diagnostic.

Namespace filters store normalized components. Matching accepts exact dot, slash, or backslash-delimited components and rejects arbitrary substrings, so rxfnsb.trace matches rxfnsb, but myrxfnsbhelper does not. Stored filters are not renormalized for each event.

_trace_escape_text scans each Unicode code point once. It escapes backslash, quote, tab, newline, form feed, carriage return, and remaining C0 controls, while appending ordinary code points directly.

7.128 Example

```
options levelb
import rxfnsb

controller = .tracecontroller()
call controller.reset_namespace_filters()
call controller.unsuppress_namespace("my.library")
call _trace_set("results")
say _trace_current_mode()       /* R */
call _trace_set("off")
```

ts_trace.crexx exercises this example and the mode, filter, escaping, signal, metadata, status, and cached-context boundaries in optimized and unoptimized forms. TRACE statement lowering and presentation remain owned by the certified compiler exit and are outside this Level B selector contract.

7.129 translate

```
translate(source = .string [, output_table = .string
          [, input_table = .string [, pad = .string]]]) = .string
```

With both tables omitted, translate applies the Level B locale-independent simple uppercase mapping. It does not normalize the text and does not perform full case folding.

With an explicit input_table, each source codepoint is replaced by the codepoint at the same position in output_table. The first duplicate input entry wins. A match beyond the output table uses `pad`; an omitted output table is an empty table. Source codepoints absent from the input table are unchanged.

When `output_table` is supplied but `input_table` is omitted, the input table is the fixed Level B U+0000 through U+00FF byte-domain range. Codepoints outside that range are unchanged. The implementation calculates those positions directly and does not allocate a 256-codepoint table.

`pad` defaults to blank and must contain exactly one Unicode codepoint; otherwise `INVALID_ARGUMENTS` is raised. Positions and table entries are codepoint based, not UTF-8 byte based.

The Classic Level C profile-dependent contract is documented separately in `lib/rxfnsc/translate.md`.

7.130 trunc

The native Level B function truncates a decimal value to fixed text:

```
trunc(number = .decimal, fraction = 0) = .string
```

`fraction` is a non-negative `.int`. The result contains exactly that many fractional digits, adding zeros when required, and is never rounded or written in exponent form. With the default zero it has no decimal point.

```
trunc(12.3)           /* "12" */
trunc(127.09782, 3)   /* "127.097" */
trunc(127.1, 3)       /* "127.100" */
trunc(1E-12, 12)      /* "0.000000000001" */
```

A negative fraction signals `INVALID_ARGUMENTS`. The input is not modified.

The implementation extracts the decimal coefficient and exponent once, removes the coefficient point, and assembles the requested fixed result with direct substring and zero-padding instructions. It performs no float conversion, power, multiplication, integer conversion, or helper call. One string copy is used to materialize the coefficient's VM text metadata before scanning it.

The Level B call boundary performs ordinary `.decimal` conversion. Invalid dynamic conversion raises the catchable `CONVERSION_ERROR` signal.

The distinct Level C `TRUNC` BIF accepts Classic numeric text through RexxValue and preserves mantissas longer than the active typed-decimal precision.

LevelB Functions

7.131 upper

`upper` returns an uppercased copy of a string:

```
upper(string = .string) = .string
```

Every character covered by the VM's deliberately limited, locale-independent simple uppercase table is converted. Other characters and an empty string are unchanged; this is not full Unicode case folding.

```
upper("MiXeD 123 !?") /* "MIXED 123 !?" */
upper("äöüé")         /* "ÄÖÜÉ" */
upper("")             /* "" */
```

U+0000 is an ordinary codepoint for this bounded operation; it does not stop conversion of later text.

The implementation does not modify the argument, and valid `.string` input has no error branch. It uses a zero-copy exposed input binding followed by one direct `strupper` instruction, creating only the result string.

This Level B helper uppercases the complete string. It does not accept start or length arguments. `UPPER` is not a required Level C BIF in the repository's Level C catalog; the common runtime's existing helper remains compatibility surface only.

7.132 value

`value(name)` is a Level B caller-frame introspection helper. It is distinct from the assigning Classic `VALUE` built-in function.

```
value(name = .string) = .string
```

The helper searches the immediate caller procedure's source metadata for a visible scalar register or constant. Names are matched case-insensitively and exactly. Integer and float registers are converted to the function's string result; string values are returned unchanged.

- An empty name returns an empty string.
- A name that is absent, cleared, or outside the immediate caller procedure returns its uppercase spelling.

- The helper is read-only and does not create or assign a variable.
- Arrays, objects, stems, and compound Classic variables are outside this Level B helper's contract.

The metadata scan stops at the caller's `.meta_func` boundary. This is required because `metalinkpreg` can safely read only a register number belonging to the immediate parent frame.

7.133 Examples

```
count = 12
say value("count")     /* 12 */
say value("missing")  /* MISSING */
say value("")          /* empty string */
```

The focused test `lib/rxfnsb/tests_functional/ts_value.crexx` covers integer, float, string, constant, blank, empty, missing, exact-name, visibility, and procedure-boundary behavior.

7.134 verify

`verify` locates the first character that either is or is not in a reference table:

```
verify(string = .string, reference = .string,
       option = "N", start = 1) = .int
```

`option` is case-insensitive and only its first character is significant:

- `N` (the default) returns the first character not in `reference`.
- `M` returns the first character in `reference`.

`start` is an optional positive 1-based integer. The result is a 1-based Unicode character position, or zero when no qualifying character exists. An empty source returns zero. With an empty reference, Nomatch returns `start` when it is within the source and Match returns zero.

```
verify("1Z3", "1234567890")          /* 2 */
verify("AB4T", "1234567890", "M")  /* 3 */
verify("1P3Q4", "1234567890",,3)   /* 4 */
verify("□éx", "□", "Match")        /* 2 */
```

An empty/invalid option or a non-positive start raises `INVALID_ARGUMENTS`. The source and reference are not modified. Their lengths are cached, and each source character is checked by one native VM character-table scan; no substring or temporary character string is allocated.

7.135 version

`version()` returns a compact description of the running VM build:

```
version() = .string
```

The result contains exactly four space-separated fields:

```
platform bits crexx-version build-date
```

- `platform` is `linux`, `windows`, `macOS`, `cms`, or `unknown`.
- `bits` is the VM pointer width, `32` or `64`.
- `crexx-version` starts with `crexx-` and may include prerelease, build-channel, commit, or dirty-worktree metadata.
- `build-date` is the VM compile date in `yyyymmdd` form.

Example shape:

```
macOS 64 crexx-1.0.0-beta.3+local.g123456789abc 20260713
```

The implementation directly executes `rxvers`; it has no arguments and does not inspect compiler or source-tree state. The instruction has no translated VM signal. Failure to allocate the returned string is fatal.

`lib/rxfnsb/tests_functional/ts_version.crexx` checks every stable field rule without hard-coding a particular release identifier.

7.136 word

```
word(string = .string, wordnum = .int) = .string
```

`word` returns the positive one-based `wordnum` word from `string`. Words are delimited by the VM Unicode whitespace definition. It returns the empty string when the requested word does not exist, including for empty or blank-only input. A non-positive word number signals `INVALID_ARGUMENTS`.

The implementation scans directly to the requested word and extracts it once; it makes no other selector calls. The focused harness is `lib/rxfnsb/tests_functional/tsword.crexx`.

7.137 wordindex

```
wordindex(string = .string, wordnum = .int) = .int
```

`wordindex` returns the positive one-based character position of `wordnum` in `string`. Words are delimited by the VM Unicode whitespace definition. It returns `0` when the requested word does not exist. A non-positive word number signals `INVALID_ARGUMENTS`.

The implementation scans only as far as the requested word and allocates no strings. The focused harness is `lib/rxfnsb/tests_functional/ts_wrdix.crexx`.

7.138 wordlength

```
wordlength(string = .string, wordnum = .int) = .int
```

`wordlength` returns the Unicode character length of the positive one-based `wordnum` in `string`. Words use the VM Unicode whitespace definition. It returns `0` when the requested word does not exist. A non-positive word number signals `INVALID_ARGUMENTS`.

The implementation scans only as far as the requested word and computes its length from source positions without allocating a substring or calling another selector. The focused harness is `lib/rxfnsb/tests_functional/ts_wordlength.crexx`.

7.139 wordpos

```
wordpos(phrase = .string, string = .string [, start = .int]) = .int
```

`wordpos` returns the positive one-based word number of the first exact word sequence matching `phrase` at or after `start`, which defaults to `1`. Whitespace at either edge is ignored and any nonempty whitespace run separates words, so different run lengths compare equally. Word text remains case- sensitive and must match exactly; prefixes are not matches.

Empty/blank-only phrases, empty targets, and absent phrases return `0`. Words use the VM Unicode whitespace definition. A non-positive start signals INVALID_-ARGUMENTS.

The implementation scans word boundaries directly and compares word spans by codepoint without allocating substrings or calling another selector. The focused harness is `lib/rxfnsb/tests_functional/ts_wordpos.crexx`.

7.140 words

```
words(string = .string) = .int
```

`words` counts the words in `string` using the VM Unicode whitespace definition. Empty and blank-only strings contain zero words.

The implementation is a single allocation-free scan and makes no other selector calls. The focused harness is `lib/rxfnsb/tests_functional/ts_words.crexx`.

7.141 x2b

```
x2b(hexadecimal = .string) = .string
```

The Level B `x2b` helper converts every hexadecimal digit to exactly four binary digits. Input is case-insensitive, the empty string returns empty, and all leading zero nibbles are retained.

Interior ASCII blanks are permitted only when an even number of hexadecimal digits lies to their right. Leading/trailing blanks, a blank followed by an odd right-hand group, or any non-hexadecimal character raises INVALID_ARGUMENTS.

```
x2b("C3")      /* "11000011" */
x2b("7")       /* "0111" */
x2b("1 C1")    /* "000111000001" */
x2b("0001")    /* "0000000000000001" */
x2b("")        /* empty */
```

The implementation performs one validation scan and one conversion scan. It uses direct character-range arithmetic and appends four table bits per digit; it does not perform whole-value numeric conversion or call another selector.

The separate Classic Level C BIF is documented in `lib/rxfnsc/x2b.md`. The focused native harness is `lib/rxfnsb/tests_functional/ts_x2b.crexx`.

7.142 x2c

```
x2c(hexadecimal = .string) = .string
```

The Level B `x2c` helper parses hexadecimal bytes and maps each byte value to the Unicode code point U+0000 through U+00FF. An odd number of digits is left-padded with one zero nibble; empty input returns empty and leading zero bytes are retained.

This produces valid Level B UTF-8 text rather than a raw byte buffer. For example, hexadecimal `FF` becomes U+00FF (UTF-8 bytes `C3 BF`), not an invalid single `FF` byte. Use `.binary` and the binary-memory helpers for untyped bytes.

Interior ASCII blanks are permitted only when an even number of hexadecimal digits lies to their right. Leading/trailing blanks, mis-grouped blanks, and non-hexadecimal characters raise `INVALID_ARGUMENTS`.

```
x2c("416263") /* "Abc" */
x2c("F")      /* U+000F */
x2c("004D")   /* U+0000 followed by "M" */
x2c("FF")     /* U+00FF */
x2c("")       /* empty */
```

The implementation validates once and converts once with direct ASCII nibble arithmetic. It does not allocate a blank-stripped copy, search conversion tables, or call another selector.

Classic Level C X2C instead uses configuration-coded characters; its separate contract and current dependency are documented in `lib/rxfnsc/x2c.md`. The focused native harness is `lib/rxfnsb/tests_functional/ts_x2c.crexx`.

7.143 x2d

```
x2d(hexadecimal = .string [, signed_length = .int]) = .int
```

The Level B `x2d` helper converts grouped hexadecimal text to an exact native signed 64-bit integer. Without `signed_length`, the text is unsigned; empty input returns zero and values above `INT64_MAX` raise `OVERFLOW_UNDERFLOW`.

With a non-negative `signed_length`, only the rightmost requested hexadecimal digits are used. Shorter input is left-padded with zero, and the selected high bit determines signed twos-complement. A zero length returns zero. Results that cannot be sign-extended into the native range raise `OVERFLOW_UNDERFLOW`.

Interior ASCII blanks are valid only when an even number of hexadecimal digits lies to their right. Invalid text or a negative length raises `INVALID_ARGUMENTS`.

```
x2d("81")        /* 129 */
x2d("81", 2)     /* -127 */
x2d("81", 4)     /* 129 */
x2d("F081", 3)   /* 129 */
x2d("F081", 2)   /* -127 */
x2d("0031", 0)   /* 0 */
```

The implementation validates/counts once and scans only the selected digits. It accepts arbitrarily many leading zeros or valid sign-extension digits without allocating a cleaned/right-aligned copy, and makes no selector call.

Classic Level C X2D returns a Rexx whole number under caller numeric settings; that separate contract is documented in `lib/rxfnsc/x2d.md`. The focused native harness is `lib/rxfnsb/tests_functional/ts_x2d.crexx`.

7.144 xrange

```
xrange(from = .string, tos = .string) = .string
```

The native Level B `xrange` helper returns the inclusive byte-domain character range from `from` through `tos`. Each endpoint must be exactly one Unicode character from U+0000 through U+00FF. The range wraps from U+00FF to U+0000 when `from` is greater than `tos`, and therefore contains between one and 256 characters.

```
xrange("a", "f")  /* "abcdef" */
```

An empty, multi-character, or out-of-domain endpoint raises `INVALID_ARGUMENTS`. The implementation directly appends at most 256 codepoints and never prints a deprecation warning. It remains a legacy byte-domain helper; use `sequence` for a non-wrapping Unicode range.

Classic Level C XRANGE is a distinct configuration-coded BIF documented in `lib/rxfnsc/xrange.md`. The focused native harness is `lib/rxfnsb/tests_functional/ts_xrange.crexx`.

8

The .stem class

The `.stem` class provides the implementation of the Classic Rexx stem variables.

Compound symbols and stems are used for more complex collections of variables [...] Compound symbols may be used to set up arrays and lists of variables, in which the subscript is not necessarily numeric, and thus offer great scope for the creative programmer. A useful application is to set up an array in which the subscripts are taken from the value of one or more variables, so effecting a form of associative memory ("content addressable").[5]

As a core language element this class is included in the namespace `rxfnsb`. In cRexx, a stem variable needs to be declared:

```
options levelb
import rxfnsb

a = .stem()
a.b = 'c'
a.c = 'd'
say 'a has' a.size() 'elements'
say 'a.b  =' a.b
say 'a[b] =' a[b]
say 'a.c  =' a.c
say 'a[c] =' a[c]
```

```
a has 2 elements
a.b  = c
a[b] = c
a.c  = d
```

[5] Cowlishaw 1985

```
a[c] = d
```

Here we can see that in addition to the . (dot) notation, CREXX supports, like Object REXX and NetRexx, the alternative square bracket notation `[x]` for stems, and that there is a method `.size()` which gives the current size of the stem variable.

Stem variables may have a composite tail, like in this example:

```
options levelb
import rxfnsb

root = .stem()
tail2 = '867_5309'
root.tail1.tail2 = 'Jenny'

say root.tail1.tail2
```

```
Jenny
```

We can loop over stem elements using the `key()` and `value()` methods, as in the next example:

```
options levelb
import rxfnsb

countries = .stem()

countries.1   = 'Germany'
countries.2   = 'The Netherlands'
countries.3   = 'The United Kingdom'

say '===================='

loop j=1 to countries.size()
  k = countries.key(j)
  v = countries.value(k)
  say k ":" v
end
```

```
===================
1 : Germany
2 : The Netherlands
3 : The United Kingdom
```

There is also support for the related `.stemIterator` class which makes *live* and *snapshot* iterators possible.

9

The .rxsocket library

`rxsocket.rexx` is the Level B wrapper around the VM's core socket instructions. It is built into `library.rxbin`.

The implementation uses the operating system socket API directly: POSIX sockets on Unix-like systems and Winsock2 on Windows. Client TLS is provided by the VM socket TLS backend when one is selected.

9.1 Handles and Status

`socketcreate()` returns a positive VM-managed socket handle[6]. The VM closes any live handles when the VM context is freed, but programs should still call `socketclose()` when finished.

Status codes are:

- `0`: success
- `1`: EOF / peer closed
- `2`: timeout
- `3`: operation would block
- negative values: invalid handle, allocation failure, OS error, DNS failure, bad argument, socket-not-open, or TLS setup/availability failure

[6] Handles are not OS file descriptors and should only be used with `rxsocket` functions.

Use `socketstatus(sock)` after operations that do not directly return a status, and `socketerror(sock)` for a short diagnostic string.

9.2 API

- `socketcreate() = .int`
- `socketclose(sock) = .int`
- `socketconnect(sock, host, port) = .int`
- `socketconnecttls(sock, host, port) = .int`
- `socketbind(sock, host, port) = .int`
- `socketlisten(sock, backlog) = .int`
- `socketaccept(sock) = .int`
- `socketshutdown(sock, how) = .int`
- `socketsend(sock, data) = .int`
- `socketsendb(sock, data) = .int`
- `socketrecv(sock, maxbytes) = .string`
- `socketrecvb(sock, maxbytes) = .binary`
- `socketpending(sock) = .int`
- `sockettimeout(sock, milliseconds) = .int`
- `socketblocking(sock, enabled) = .int`
- `socketnodelay(sock, enabled) = .int`
- `socketkeepalive(sock, enabled) = .int`
- `socketpeer(sock) = .string`
- `socketlocal(sock) = .string`
- `socketstatus(sock) = .int`
- `socketerror(sock) = .string`

`socketshutdown()` uses the portable mode values used by the VM instruction: `0` for receive, `1` for send, and `2` for both.

`socketconnecttls()` connects to `host:port` and starts client TLS before any application bytes are exchanged, using `host` for SNI and certificate hostname verification. After a successful secure connect, the existing `socketsend()` and `socketrecv()` calls operate over the encrypted stream. In builds without a TLS backend, the instruction still exists and fails with a negative status so programs can probe capability with `socketstatus()` / `socketerror()`.

The public Level B wrapper does not expose true STARTTLS. A lower-level RXAS instruction exists for future protocol-specific libraries that need to exchange clear-text bytes before TLS, but backends may return unsupported if they cannot upgrade an existing connection in place.

TLS is selected at CMake configure time. Fresh build directories default to `NETWORK` on Apple platforms, `OPENSSL` on non-Windows Unix-like platforms, and `SCHANNEL` on Windows:

```
cmake -S . -B cmake-build-tls -DCREXX_ENABLE_TLS=OPENSSL
cmake -S . -B cmake-build-tls-network -DCREXX_ENABLE_TLS=NETWORK
cmake -S . -B cmake-build-tls-windows -DCREXX_ENABLE_TLS=SCHANNEL
cmake -S . -B cmake-build-notls -DCREXX_ENABLE_TLS=OFF
```

`NETWORK` is the macOS backend. It links against Apple's Network.framework, Security.framework, and CoreFoundation.framework, and uses the operating system trust store, so the VM binaries do not acquire an OpenSSL runtime dependency. It implements `socketconnecttls()` as a native secure connection from byte zero. The lower-level true STARTTLS instruction reports unsupported on this backend.

`OPENSSL` is the portable backend for platforms where system TLS is not wired in yet, such as Linux. It uses OpenSSL's default verification paths and performs hostname verification.

`SCHANNEL` is the Windows backend. It uses SChannel/SSPI with the Windows trust store and hostname verification, so Windows VM binaries do not need OpenSSL for core socket TLS.

Use `-DCREXX_TLS_STATIC_OPENSSL=ON` when the OpenSSL package and platform toolchain support static linking. This option only applies to `CREXX_ENABLE_TLS=OPENSSL`.

`socketlocal()` and `socketpeer()` return numeric endpoint text in `host:port` form. IPv6 addresses are currently emitted in the same simple form, so higher-level parsers should treat the final colon-separated field as the port if they need to split it.

9.3 Loopback Example

```
options levelb
import rxsocket

host = "127.0.0.1"
port = 32164

server = socketcreate()
if socketbind(server, host, port) <> 0 then exit 1
if socketlisten(server, 5) <> 0 then exit 1

client = socketcreate()
if socketconnect(client, host, port) <> 0 then exit 1

accepted = socketaccept(server)
if accepted <= 0 then exit 1

if socketsend(client, "ping") <> 4 then exit 1
if socketrecv(accepted, 4) <> "ping" then exit 1

drop_rc = socketclose(client)
drop_rc = socketclose(accepted)
drop_rc = socketclose(server)
```

The functional coverage lives in `lib/rxfnsb/tests_functional/ts_rxsocket.rexx`. `lib/rxfnsb/tests_functional/ts_socket_tls_live.rexx` is a default-skipped live TLS handshake smoke test; set `CREXX_TLS_LIVE_SMOKE=1` to enable it on a runner with network access. The lower-level RXAS instruction coverage lives in `interpreter/tests/tests_socket.rxas`.

9.4 Echo server example

The below sample implements an echo-type server to illustrate the flow over the network. In the example, the `netcat` utility (abbreviated as a command to `nc`) is used to send input over a socket on port 10001 (on the same machine) to the server, which will promptly echo it back to the shell in which `netcat` is running.

```
options levelb comments_dash

import rxsocket
import rxfnsb

host     = "127.0.0.1"
port     = 10001
backlog     = 5
bufferSize = 4096
```

```
server = socketcreate()
if server <= 0 then exit 1

if socketbind(server, host, port) <> 0 then do
  drop_rc = socketclose(server)
  exit 1
end

if socketlisten(server, backlog) <> 0 then do
  drop_rc = socketclose(server)
  exit 1
end

say "Backend server listening on "host":"port

do forever
  client = socketaccept(server)

  if client <= 0 then do
    say "socketaccept failed"
    leave
  end

  say "Client connected"

  do forever
    data = socketrecv(client, bufferSize)
    dl = length(data)

    if left(data,dl-1) = 'version' then data=version()
    if data = "" then leave

    bytesSent = socketsend(client, data)
    say bytesSent 'bytes send'
  end -- do forever

  drop_rc = socketclose(client)
  say "Client disconnected"
end

drop_rc = socketclose(server)
```

Run this, and then use `nc` to send input to it:

```
nc 127.0.0.1 10001
```

The server listens on port 10001 of localhost and will echo everything that is typed into `netcat` back to that session; except for when the string is `version`, in which case it will return the version of the running CREXX runtime. The comparision needs to be to one character less of the `data` variable, because this will include a linefeed.

10

The .rxjson library

`rxjson` is the Level B strict JSON foundation library. It is implemented in CREXX in `lib/rxfnsb/rexx/rxjson.crexx`, is compiled into `library.rxbin`, and does not require a native plugin.

The legacy functions remain string-oriented. Code that traverses the same JSON more than once can instead construct one immutable `.jsondocument` and reuse its private structural index.

```
options levelb
import rxjson

response = '{"choices":[{"message":{"content":"Hello"}}]}'
say jsonget(response, "choices.1.message.content")
```

10.1 Import

```
import rxjson
```

The module exposes:

- `.jsondocument(json)`
- `jsonvalid(json)`
- `jsontype(json, path)`
- `jsonget(json, path)`
- `jsoncount(json, path)`

- `jsonmembers(json, path, names[])`
- `jsonscancontainer(text, from, document, start, after)`
- `jsonquote(text)`
- `jsonunquote(json)`
- `jsonarray(values[])`
- `jsonobject(keys[], values[])`

10.2 Value Model

The legacy functions accept and return JSON values as strings. A `.jsondocument` owns the exact source string supplied to its constructor and a private index of every value. The source cannot be changed through the public API. Node identifiers are positive, document-local and ephemeral: never retain them across documents or treat them as an ABI or serialized format. Zero means that lookup found no node.

`jsonget()` returns REXX strings:

- JSON strings are unquoted and unescaped.
- JSON numbers are returned as their JSON text.
- JSON booleans are returned as `true` or `false`.
- JSON null is returned as `null`.
- JSON arrays and objects are returned as their JSON substring.
- Missing paths and invalid JSON return the empty string.

Use `jsontype()` when an empty result could be ambiguous.

10.3 Path Syntax

Paths select values inside JSON objects and arrays.

- `""` selects the root value.
- Object keys are separated with dots, for example `usage.total_tokens`.
- Array indexes are one-based, for example `choices.1.message.content`.
- Bracket array indexes are also supported, for example `choices[1].finish_reason`.

- Object keys are case-sensitive.

The simple path syntax is intentionally small. Keys containing dots or square brackets are not addressable through paths. Use `member(parent, name)` for arbitrary object keys.

10.4 Immutable Document

Constructing a document validates one complete strict JSON value and builds its index once:

```
document = .jsondocument('{"items":[{"name":"first"}]}')
if document.valid() then do
  items = document.member(document.root(), "items")
  first = document.element(items, 1)
  say document.node_get(document.member(first, "name"))
end
```

Document status and storage methods are:

Method	Contract
`valid()`	`1` for one complete strict JSON value, otherwise `0`.
`error_code()`	`none`, `syntax`, `truncated`, or `trailing`.
`error_position()`	One-based character position, or `0` when valid.
`error()`	Human-readable diagnostic; do not parse it as a stable code.
`source()`	The immutable source owned by the document.
`source_bytes()`	UTF-8 source size in bytes.
`index_bytes()`	Current private index footprint in bytes.
`node_count()`	Total number of indexed JSON values.
`node_count(node)`	Immediate child count, `0` for a scalar, or `-1` for an invalid node.

The compatibility methods `type(path)`, `get(path)`, `count(path)`, and `members(path, names[])` have the same results as their `json*` function counterparts. `find(path)` returns a positive node identifier, `0` for a missing path, or `-1` for an invalid document or path.

Use node traversal when a key cannot be represented by the legacy path syntax or when repeated traversal should avoid reparsing:

Method	Contract
`root()`	Root node, or `0` for an invalid document.
`member(parent, name)`	Exact decoded object-key lookup: positive node, `0` missing, `-1` invalid node/document, `-2` wrong container type.
`element(parent, index)`	One-based array lookup with the same status convention as `member()`.
`node_type(node)`	`object`, `array`, `string`, `number`, `boolean`, `null`, or `error`.
`node_get(node)`	Decoded string value or exact JSON source text; empty for an invalid node.
`children(node, nodes[])`	Replaces `nodes[]` with immediate child IDs in source order; returns their count, `0` for a scalar, or `-1` for invalid input.
`node_name(node)`	Decoded object key or one-based array index; empty for a root or invalid node.

Typed getters write through an exposed result and return a status. They never coerce JSON null or a different JSON type:

Method	Success value	Additional failures
`node_string(node, value)`	Decoded `.string`	`-1` invalid, `-2` wrong type.
`node_boolean(node, value)`	`.int` `0` or `1`	`-1` invalid, `-2` wrong type.
`node_int(node, value)`	Exact signed 64-bit `.int`; integral forms such as `1.0` and `1e3` are accepted.	`-1` invalid, `-2` wrong type, `-3` non-integral, `-4` range.
`node_float(node, value)`	Zero or a finite normal binary64 `.float`	`-1` invalid, `-2` wrong type, `-4` overflow or underflow, including a nonzero subnormal result.

10.4.1 Explicit packed numeric projection

Numeric JSON arrays remain ordinary JSON arrays. A caller that explicitly chooses a numeric width may project an indexed array directly to an owning, headerless, canonical-little-endian `.binary`:

Method	Successful output
`node_f32_array(node, expected_count, packed, error)`	Exactly `count * 4` bytes containing IEEE binary32 values.
`node_i64_array(node, expected_count, packed, error)`	Exactly `count * 8` bytes containing signed 64-bit integers.

`packed` and `error` are exposed outputs. `expected_count` is `-1` to accept the JSON array's count, or a required non-negative count. Success returns `0`, replaces `packed`, and clears `error`. Failure clears `packed`, writes a diagnostic, and returns `-1` for an invalid document/node, `-2` for a non-array node, `-3` for an invalid expected count, `-4` for a count mismatch, `-5` for a non-number element, or `-6` for conversion/range failure.

The f32 projection rejects overflow and any nonzero JSON value that underflows to stored zero; conversion signals are reported as status `-6` rather than escaping the method. The i64 projection accepts exact integral decimal and exponent forms such as `1.0` and `1e3`, and rejects non-integral or out-of-range values. Empty arrays produce empty binaries when the count policy permits. There is no type inference, header, magic, version, normalization flag or hidden cache. Persisted users must retain the chosen type and count in their own schema.

10.5 API Reference

10.5.1 `jsonvalid(json) = .int`

Returns `1` when `json` is a complete valid JSON value, otherwise `0`.

```
ok = jsonvalid('{"ready":true}')
```

10.5.2 `jsontype(json, path) = .string`

Returns the type at `path`.

Possible return values are:

- `object`
- `array`
- `string`

- `number`
- `boolean`
- `null`
- `missing`
- `error`

`missing` means the JSON was valid but the path did not resolve. `error` means the JSON or path expression could not be parsed.

```
kind = jsontype('{"ready":true}', "ready")
```

10.5.3 `jsonget(json, path) = .string`

Returns the value at `path` as a REXX string. Strings are unquoted; scalar JSON values are returned as their JSON spelling; object and array values are returned as JSON text.

```
content = jsonget(response, "choices.1.message.content")
tokens = jsonget(response, "usage.total_tokens")
```

10.5.4 `jsoncount(json, path) = .int`

Returns the number of members in an object or elements in an array.

- Returns `0` for scalar values.
- Returns `-1` for invalid JSON or missing paths.

```
count = jsoncount(response, "choices")
```

10.5.5 `jsonmembers(json, path, names[]) = .int`

Writes member names or array indexes into `names[]` and returns the count.

- Object members write their object keys.
- Array members write one-based indexes as strings.
- Returns `-1` for invalid JSON or missing paths.

Pass a fresh `.string[]` when possible. The function writes `names[1]` through `names[count]`; it does not clear older entries beyond the returned count.

```
names = .string[]
count = jsonmembers(response, "", names)
do i = 1 to count
  say names[i]
end
```

10.5.6 **jsonscancontainer(text, from, document, start, after) = .int**

Finds the first complete strict JSON object or array whose opening delimiter is at or after the one-based character position `from`. On success it returns `1`, writes an already indexed `.jsondocument`, and writes one-based character positions `start` (inclusive) and `after` (one past the value). Pass `after` as the next `from` value to continue scanning. Values that are only JSON scalars are not candidates.

On failure it returns `0`, sets `start` and `after` to zero, and returns an invalid empty document. Values below `1` are normalized to `1`; a position past the input returns no match. The scanner identifies structural JSON only. It does not decide whether a found value satisfies an application contract, so a caller may validate the document and continue from `after` when necessary.

```
text = 'prefix {"kind":"first"} then {"kind":"second"}'
document = .jsondocument("")
start = 0
after = 0
if jsonscancontainer(text, 1, document, start, after) then do
  say document.get("kind")
  if jsonscancontainer(text, after, document, start, after) then
    say document.get("kind")
end
```

The successful-slice construction deliberately uses two grammar passes: one allocation-free boundary-validation pass over the original input, followed by one indexing pass that owns the exact slice and builds its sole index. Rejected candidate openers construct no candidate document or index; the public call initializes its output to one invalid empty document so failure has a deterministic result. This lifecycle avoids copying every candidate while keeping `.jsondocument` construction and ownership uniform.

10.5.7 **jsonquote(text) = .string**

Returns `text` as a JSON string literal, including surrounding quotes.

```
name = jsonquote('Ada "Lovelace"')
```

10.5.8 jsonunquote(json) = .string

Returns the REXX string represented by a JSON string literal. If `json` is not a valid JSON string, returns the empty string.

```
text = jsonunquote('"hello \"json\""')
```

10.5.9 jsonarray(values[]) = .string

Builds a JSON array from an array of already encoded JSON values. Returns the empty string if any element is not valid JSON.

```
values = .string[]
values[1] = jsonquote("system")
values[2] = jsonquote("hello")
roles = jsonarray(values)
```

10.5.10 jsonobject(keys[], values[]) = .string

Builds a JSON object from string keys and already encoded JSON values. Keys are quoted by `jsonobject()`. Values must already be valid JSON. Returns the empty string when key and value counts differ or any value is invalid JSON.

```
keys = .string[]
values = .string[]
keys[1] = "role"
values[1] = jsonquote("user")
keys[2] = "content"
values[2] = jsonquote("Say hi")
message = jsonobject(keys, values)
```

10.6 Constructing JSON Example

This example builds a small nested JSON value from already encoded values.

```
options levelb
import rxjson

item_keys = .string[]
```

```
item_values = .string[]
item_keys[1] = "name"
item_values[1] = jsonquote("sample")
item_keys[2] = "enabled"
item_values[2] = "true"
item = jsonobject(item_keys, item_values)

items = .string[]
items[1] = item

body_keys = .string[]
body_values = .string[]
body_keys[1] = "version"
body_values[1] = "1"
body_keys[2] = "items"
body_values[2] = jsonarray(items)

body = jsonobject(body_keys, body_values)
```

10.7 Current Limits

- `rxjson` is not a schema validator or repair engine. Required fields, additional fields, nullability, and application-specific selection remain caller policy.
- Paths cannot address object keys that contain dots or square brackets; use document node traversal for those keys.
- Object key matching is case-sensitive.
- Unicode `\uXXXX` unquoting decodes Unicode code points and paired UTF-16 surrogate escapes; malformed or unpaired surrogate escapes are rejected.
- Duplicate object key handling follows first-match lookup behaviour today; do not depend on duplicates in new data.
- JSON numeric arrays remain ordinary JSON arrays. The library does not infer integer widths, floating-point widths, byte order, or packed storage.
- The index representation and node identifiers are private and may change between builds.

10.8 Tests

Functional coverage lives in:

- `lib/rxfnsb/tests_functional/ts_rxjson.crexx`
- `lib/rxfnsb/tests_functional/ts_rxjson_document.crexx`
- `lib/rxfnsb/tests_functional/ts_rxjson_noisy_contract.crexx`

The focused test target is:

```
ctest --test-dir cmake-build-debug -R '^ts_rxjson' --output-on-failure
```

11

The .rexx class

This section describes the set of methods defined for the Rexx class, a container which enables object-oriented string manipulation. These methods implement the traditional *built-in functions*, and include character manipulation, word manipulation, conversion, and arithmetic functions.

Use of these methods needs import of the rexx package:

```
import rexx
```

Rexx strings must be constructed with a factory constructor.

General notes on the built-in methods:

- Every method operates on the receiver's native `.string` value; this is referred to by the name *string* in the descriptions. For example, if the **word** method were invoked using the term:

  ```
  v=.rexx("Three word phrase"); v.word(2)
  ```

  word the name *string* refers to the string
  "**Three word phrase**", and the name *n* refers to the integer **2**.
- Method arguments use the native types shown by their signatures: positions, lengths, counts, and widths are `.int`; text and pad arguments are `.string`. Transformations normally return `.rexx` for fluent chaining, while observations, positions, counts, and predicates return native scalar types. `stringValue()` or `toString()` extracts the wrapped string.
- The first parenthesis in a method call must immediately follow the name of the method, with no space in between.

- A position in a string is the number of a character in the string, where the first character is at position 1, *etc.*
- A *pad* argument, if specified, must be exactly one character long.
- If a method has a sub-option selected by the first character of a string, that character may be in upper or lowercase.
- Conversion between character encodings and decimal or hexadecimal is dependent on the machine representation (encoding) of characters and hence will return appropriately different results for Unicode, ASCII, and EBCDIC.

11.1 abbrev(info [,length])

returns 1 if *info* is equal to the leading characters of *string* and *info* is not less than the minimum length, *length*; 0 is returned if either of these conditions is not met. *length* must be a non-negative whole number; the default is the length of *info*.
Examples:

```
v=.rexx('Print'); v.abbrev('Pri')   == 1
w=.rexx('PRINT'); w.abbrev('Pri')   == 0
w=.rexx('PRINT'); w.abbrev('PRI',4) == 0
w=.rexx('PRINT'); w.abbrev('PRY')   == 0
w=.rexx('PRINT'); w.abbrev('')      == 1
w=.rexx('PRINT'); w.abbrev('',1)    == 0
```

Note: A null string will always match if a length of 0 (or the default) is used. This allows a default keyword to be selected automatically if desired. **Example:**

```
say 'Enter option:';  option=ask
select  /* keyword1 is to be the default */
  v=.rexx('keyword1'); when v.abbrev(option) then ...
  w=.rexx('keyword2'); when w.abbrev(option) then ...
     ...
  otherwise ...
  end
```

11.2 abs()

returns the absolute value of *string*, which must be a number. Any sign is removed from the number, and it is then formatted by adding zero with a digits setting that is either nine or, if greater, the number of digits in the mantissa of the number (excluding leading insignificant zeros). Scientific notation is used, if necessary.

```
v=.rexx('12.3'); v.abs                == 12.3
w=.rexx(' -0.307'); w.abs             == 0.307
x=.rexx('123.45E+16'); x.abs          == 1.2345E+18
y=.rexx('- 1234567.7654321'); y.abs == 1234567.7654321
```

The method accepts leading/trailing Unicode whitespace and whitespace between an initial sign and the numeric text. Other invalid numeric text raises `CONVERSION_ERROR`. The typed Level B function and Classic Level C BIF remain separate contracts: `abs.md` and `rxfnsc/abs.md`.

11.3 b2d()

Binary to decimal. Converts the receiver's binary digits to a non-negative Level B integer. The empty string returns 0. Standard nibble-group blanks are accepted; invalid text signals `INVALID_ARGUMENTS`. Results must fit the signed 64-bit `.int` range or `OVERFLOW_UNDERFLOW` is signalled.

```
v=.rexx('01110'); v.b2d == 14
w=.rexx('10000001'); w.b2d == 129
x=.rexx('111110000001'); x.b2d == 3969
y=.rexx('1111111110000001'); y.b2d == 65409
z=.rexx('1100011011110000'); z.b2d == 50928
```

The callable has no signed-width argument. See the stable Level B `b2d` contract.

11.4 b2x()

Binary to hexadecimal. Converts *string*, a string of at least one binary (**0** and/or **1**) digits, to an equivalent string of hexadecimal characters. The returned string will use uppercase Roman letters for the values A-F, and will not include any blanks. If the number of binary digits in the string is not a multiple of four, then up to three **'0'** digits will be added on the left before conversion to make a total that is a multiple of four.

```
v=.rexx('11000011'); v.b2x  == 'C3'
w=.rexx('10111'); w.b2x     == '17'
x=.rexx('0101'); x.b2x      == '5'
y=.rexx('101'); y.b2x       == '5'
```

```
z=.rexx('111110000'); z.b2x == '1F0'
```

11.5 center(length [,pad])

or

11.6 centre(length [,pad])

returns a string of length *length* with *string* centered in it, with *pad* characters added as necessary to make up the required length. *length* must be a non-negative whole number. The default *pad* character is blank. If the string is longer than *length*, it will be truncated at both ends to fit. If an odd number of characters are truncated or added, the right hand end loses or gains one more character than the left hand end.

```
v=.rexx('ABC'); v.centre(7)          == '  ABC  '
v=.rexx('ABC'); v.center(8,'-')      == '--ABC---'
w=.rexx('The blue sky'); w.centre(8) == 'e blue s'
w=.rexx('The blue sky'); w.center(7) == 'e blue '
```

Note: This method may be called either **centre** or **center**, which avoids difficulties due to the difference between the British and American spellings.

11.7 changestr(needle, new)

returns a copy of *string* in which each occurrence of the *needle* string is replaced by the *new* string. Each unique (non-overlapping) occurrence of the *needle* string is changed, searching from left to right and starting from the first (leftmost) position in *string*. Only the original *string* is searched for the *needle*, and each character in *string* can only be included in one match of the *needle*.

If the *needle* is the null string, the result is a copy of *string*, unchanged.

```
v=.rexx('elephant'); v.changestr('e','X')    == 'XlXphant'
v=.rexx('elephant'); v.changestr('ph','X')   == 'eleXant'
v=.rexx('elephant'); v.changestr('ph','hph') == 'elehphant'
v=.rexx('elephant'); v.changestr('e','')     == 'lphant'
v=.rexx('elephant'); v.changestr('','!!')    == 'elephant'
```

11.8 compare(target [,pad])

returns 0 if *string* and *target* are the same. If they are not, the returned number is positive and is the position of the first character that is not the same in both strings. If one string is shorter than the other, one or more *pad* characters are added on the right to make it the same length for the comparison. The default *pad* character is a blank.

```
v=.rexx('abc'); v.compare('abc')       == 0
v=.rexx('abc'); v.compare('ak')        == 2
w=.rexx('ab '); w.compare('ab')        == 0
w=.rexx('ab '); w.compare('ab',' ')    == 0
w=.rexx('ab '); w.compare('ab','x')    == 3
x=.rexx('ab-- '); x.compare('ab','-') == 5
```

11.9 copies(n)

returns *n* directly concatenated copies of *string*. *n* must be positive or 0; if 0, the null string is returned.

```
v=.rexx('abc'); v.copies(3) == 'abcabcabc'
v=.rexx('abc'); v.copies(0) == ''
w=.rexx(''); w.copies(2)    == ''
```

11.10 countstr(needle)

returns the count of non-overlapping occurrences of the *needle* string in *string*, searching from left to right and starting from the first (leftmost) position in *string*.

If the *needle* is the null string, **0** is returned.

```
v=.rexx('elephant'); v.countstr('e')  == '2'
v=.rexx('elephant'); v.countstr('ph') == '1'
v=.rexx('elephant'); v.countstr('')   == '0'
```

The **changestr** method can be used to change occurrences of *needle* to some other string.

11.11 c2d()

Coded character to decimal. Converts the Unicode code point of the character in *string* (which must be exactly one character) to its native `.int` representation. Empty or multi-character receivers raise `CONVERSION_ERROR`.

```
v=.rexx('M'); v.c2d  == 77
w=.rexx('□'); w.c2d  == 945
x=.rexx('□'); x.c2d == 128293
y=.rexx('00'x); y.c2d == 0
```

The **c2x** method can be used to convert the encoding of a character to a hexadecimal representation.

11.12 c2x()

Coded characters to hexadecimal. Converts every character in the receiver to two uppercase hexadecimal digits. The empty receiver returns an empty Rexx string; multi-character receivers are valid and leading zero digits are retained. The method uses the native Level B C2X contract, including its established low-byte behavior for Unicode code points beyond one byte.

```
v=.rexx('M'); v.c2x     == '4D'
w=.rexx('72s'); w.c2x   == '373273' -- ASCII/Unicode build
x=.rexx('0123'x); x.c2x == '0123'
y=.rexx(''); y.c2x      == ''
```

c2d method can be used to convert the encoding of a character to a decimal number.

11.13 datatype(option)

returns 1 if *string* matches the description requested with the *option*, or 0 otherwise. With the option omitted it returns `NUM` or `CHAR`. The null string is valid for `B` and `X` and false for the other explicit tests.

Only the first character of *option* is significant, and it may be in either uppercase or lowercase. The following *option* characters are recognized:

A (Alphanumeric); returns 1 if *string* only contains characters from the ranges "a-z", "A-Z", and "0-9".

B (Binary); returns 1 if *string* only contains the characters "0" and/or "1".

D (Digits); returns 1 if *string* only contains characters from the range "0-9". This is the CREXX Level B extension and is not a Classic Level C option.

L (Lowercase); returns 1 if *string* only contains characters from the range "a-z".

M (Mixed case); returns 1 if *string* only contains characters from the ranges "a-z" and "A-Z".

N (Number); returns 1 if *string* is a syntactically valid number that could be added to **'0'** without error,

S (Symbol); returns 1 if *string* only contains characters that are valid in non-numeric symbols (the alphanumeric characters and underscore), and does not start with a digit. Note that both uppercase and lowercase letters are permitted.

U (Uppercase); returns 1 if *string* only contains characters from the range "A-Z".

W (Whole Number); returns 1 if *string* is a syntactically valid number that can be added to **'0'** without error, and whose decimal part after that addition, with no rounding, is zero.

X (heXadecimal); returns 1 if *string* only contains characters from the ranges "a-f", "A-F", and "0-9".

```
v=.rexx('101'); v.datatype('B')     == 1
w=.rexx('12.3'); w.datatype('D')    == 0
w=.rexx('12.3'); w.datatype('N')    == 1
w=.rexx('12.3'); w.datatype('W')    == 0
x=.rexx('LaArca'); x.datatype('M') == 1
y=.rexx(''); y.datatype('M')        == 0
z=.rexx('Llanes'); z.datatype('L') == 0
v1=.rexx('3 d'); v1.datatype('s')     == 0
v2=.rexx('BCd3'); v2.datatype('X')    == 1
v3=.rexx('BCgd3'); v3.datatype('X')   == 0
```

Note: The Level B method traverses codepoints safely but deliberately uses the listed ASCII character classes. Whole-number testing is exact and does not use a floating-point tolerance. Explicit empty or unsupported options signal `INVALID_ARGUMENTS`.

11.14 delstr(n [,length])

returns a copy of *string* with the sub-string of *string* that begins at the *nth* character, and is of length *length* characters, deleted. If *length* is not specified, or is greater than the number of characters from *n* to the end of the string, the rest of the string is deleted (including the *nth* character). *length* must be a non-negative whole number, and *n* must be a positive whole number. If *n* is greater than the length of *string*, the string is returned unchanged. An explicitly supplied zero length also returns the string unchanged. Invalid values raise `INVALID_ARGUMENTS`.

```
v=.rexx('abcd'); v.delstr(3)     == 'ab'
w=.rexx('abcde'); w.delstr(3,2) == 'abe'
w=.rexx('abcde'); w.delstr(6)   == 'abcde'
```

See the typed `delstr` contract and the separate Classic `rxfnsc/delstr` contract.

11.15 delword(n [,length])

returns a copy of *string* with the sub-string of *string* that starts at the *nth* word, and is of length *length* blank-delimited words, deleted. If *length* is not specified, or is greater than number of remaining words in the string, it defaults to be the remaining words in the string (including the *nth* word). *length* must be a non-negative whole number, and *n* must be a positive whole number. If *n* is greater than the number of words in *string*, the string is returned unchanged. The string deleted includes any blanks following the final word involved, but none of the blanks preceding the first word involved.

```
v=.rexx('Now is the  time'); v.delword(2,2) == 'Now time'
w=.rexx('Now is the time '); w.delword(3)   == 'Now is '
x=.rexx('Now  time'); x.delword(5)          == 'Now  time'
```

11.16 d2b()

Returns the minimal string of `0` and `1` characters representing the receiver's non-negative native integer value. Zero returns `"0"`; other results have no leading zeroes. A negative receiver raises `INVALID_ARGUMENTS`. The method has no width argument and does not perform signed extension or truncation.

```
v=.rexx('0'); v.d2b     == '0'
w=.rexx('9'); w.d2b     == '1001'
x=.rexx('19'); x.d2b    == '10011'
y=.rexx('129'); y.d2b   == '10000001'
```

The native function contract is documented in `d2b.md`.

11.17 d2c()

Converts the receiver's whole-number text to one Unicode character. The value must name a Unicode scalar from `0` through `1114111` (`U+10FFFF`), excluding the surrogate range. Invalid values raise `CONVERSION_ERROR`.

The `.Rexx.d2c()` method has no length argument. The underlying typed Level B function also exposes a separate optional zero-or-one output length; callers that need that native surface should call `rxfnsb.d2c` directly.

```
v=.rexx('77'); v.d2c().toString() == 'M'
w=.rexx('+77'); w.d2c().toString() == 'M'
x=.rexx('945'); x.d2c().toString() == '□'
y=.rexx('128293'); y.d2c().toString() == '□'
```

The native function contract is documented in `d2c.md`. Classic Level C D2C is a different configuration-coded BIF documented in `rxfnsc/d2c.md`.

11.18 d2x()

Converts the receiver's non-negative native-integer text to minimal uppercase hexadecimal. Zero returns `"0"`; a negative receiver raises `INVALID_ARGUMENTS` because the method has no width from which to derive a signed representation.

The `.Rexx.d2x()` method accepts no arguments. The underlying typed Level B function also exposes an optional exact output width; callers needing padding, truncation, or signed twos-complement should call `rxfnsb.d2x` directly.

```
v=.rexx('0'); v.d2x().toString() == '0'
w=.rexx('9'); w.d2x().toString() == '9'
x=.rexx('129'); x.d2x().toString() == '81'
```

The native function contract is documented in `d2x.md`. Classic Level C D2X accepts arbitrary caller-context whole numbers and is documented in `rxfnsc/d2x.md`.

11.19 exists(index)

returns 1 if *index* names a sub-value of *string* that has explicitly been assigned a value, or 0 otherwise.

Example: Following the instructions:

```
vowel=0
vowel['a']=1
vowel['b']=1
vowel['b']=null -- drops previous assignment
```

then:

```
vowel.exists('a') == '1'
vowel.exists('b') == '0'
vowel.exists('c') == '0'
```

11.20 format([before [,after [,expp [,expt]]]])

Formats the receiver as a decimal using the caller's `NUMERIC DIGITS` and `NUMERIC FORM`. Leading and trailing whitespace, including whitespace between an initial sign and the numeric text, is accepted. Other invalid numeric text raises `CONVERSION_ERROR`.

All four controls are optional non-negative integers and omission is significant. Use an omitted argument slot, not `null`, when supplying an option to its right.

- *before* is the width of the integer part, including a possible minus sign.
- *after* is the exact number of fractional digits; the value is rounded and zero-filled to fit.
- *expp* is the exponent digit width. An explicit zero suppresses exponent notation; a nonzero width leaves a blank exponent field when the exponent is zero.
- *expt* is the trigger for exponential notation. When omitted while *expp* is present, the current `NUMERIC DIGITS` value is used.

The current `NUMERIC FORM` setting selects scientific or engineering layout; there is no separate `exform` method argument. A negative control or a field that cannot fit raises `INVALID_ARGUMENTS`.

```
v=.rexx(' - 12.73'); v.format().toString()       == '-12.73'
w=.rexx('1.75'); w.format(4,1).toString()        == '   1.8'
x=.rexx('12345.73'); x.format(,,2,2).toString()  == '1.234573E+04'
y=.rexx('12345.73'); y.format(,3,,0).toString()  == '1.235E+4'
```

The typed formatter is documented in `format.md`. Classic Level C FORMAT has the separate RexxValue contract in `rxfnsc/format.md`.

11.21 insert(new [,n [,length [,pad]]])

inserts the string *new*, padded or truncated to length *length*, into a copy of the target *string* after the *nth* character; the string with any inserts is returned. *length* and *n* must be a non-negative whole numbers. If *n* is greater than the length of the target string, padding is added before the *new* string also. The default value for *n* is 0, which means insert before the beginning of the string. The default value for *length* is the length of *new*. The default *pad* character is a blank. Invalid positions, lengths, or pads raise `INVALID_ARGUMENTS`.

```
v=.rexx('abc'); v.insert('123')         == '123abc'
w=.rexx('abcdef'); w.insert(' ',3)      == 'abc def'
v=.rexx('abc'); v.insert('123',5,6)     == 'abc  123   '
v=.rexx('abc'); v.insert('123',5,6,'+') == 'abc++123+++'
v=.rexx('abc'); v.insert('123',0,5,'-') == '123--abc'
```

The receiver is always the target and *new* is the first method argument. See the typed `insert` contract and separate Classic `rxfnsc/insert` contract.

11.22 lastpos(needle [,start])

returns the position of the last occurrence of the string *needle* in *string* (the ”haystack”), searching from right to left. If the string *needle* is not found, or is the null string, 0 is returned. By default the search starts at the last character of *string* and scans backwards. This may be overridden by specifying *start*, the point at which to start the backwards scan. *start* must be a positive whole number, and defaults to the value *string*.**length** if larger than that value or if not specified (with a minimum default value of one). An explicitly supplied zero or negative *start* raises `INVALID_ARGUMENTS`.

```
v=.rexx('abc def ghi'); v.lastpos(' ')   == 8
```

```
v=.rexx('abc def ghi'); v.lastpos(' ',7) == 4
w=.rexx('abcdefghi'); w.lastpos(' ')     == 0
w=.rexx('abcdefghi'); w.lastpos('cd')    == 3
x=.rexx(''); x.lastpos('?')              == 0
```

See the typed `lastpos` contract and the separate Classic `rxfnsc/lastpos` contract.

11.23 left(length [,pad])

returns a string of length *length* containing the left-most *length* characters of *string*. The string is padded with *pad* characters (or truncated) on the right as needed. The default *pad* character is a blank. *length* must be a non-negative whole number. This method is exactly equivalent to *string*.**substr(1**, *length* [, *pad*]**)**. Invalid lengths or pads raise `INVALID_ARGUMENTS`.

```
v=.rexx('abc d'); v.left(8)     == 'abc d   '
v=.rexx('abc d'); v.left(8,'.') == 'abc d...'
w=.rexx('abc defg'); w.left(6)  == 'abc de'
```

See the typed `left` contract.

11.24 length()

returns the number of characters in *string*.

```
v=.rexx('abcdefgh'); v.length == 8
w=.rexx(''); w.length         == 0
```

11.25 linein(name)

reads a UTF text line from the stream named by the first argument and returns it without the line terminator. A final line terminator at physical end-of-file does not create an extra empty record, although physical blank lines are preserved. This is a Level B `.string` text function, not a binary byte input function. The companion `lines(name)` predicate returns `-1` when the stream cannot be opened, `0` at end of file, and `1` when a record can be read.

11.26 lineout(name,string)

returns 0 after writing the second argument as UTF text followed by a newline to the stream named by the first argument. When the second argument is omitted, the named stream is closed. This is a Level B `.string` text function, not a binary byte output function. Separate binary file BIFs use `.binary` values.

11.27 readbinary(path)

Reads the complete named file and returns its exact contents as `.binary`. Embedded NUL and byte sequences that are not valid UTF-8 are preserved. The operation raises `NOTREADY` if the file cannot be opened, read, or closed. This is a whole-file convenience; callers remain responsible for the memory cost of the returned value.

11.28 writebinary(path,data)

Replaces the named file with the exact bytes in the `.binary` second argument and returns the number of bytes written. An empty value creates or truncates a zero-byte file. The operation raises `NOTREADY` if the file cannot be opened, written, flushed, or closed.

11.29 lower()

Returns a copy of the complete receiver with its Unicode uppercase characters converted to lowercase. The `.Rexx` method has no start or length extension.

```
v=.rexx('SumA'); v.lower().toString() == 'suma'
w=.rexx('ÉCOLE'); w.lower().toString() == 'école'
x=.rexx(''); x.lower().toString() == ''
```

See the native `lower` contract.

11.30 max(number)

returns the larger of *string* and *number*, which must both be numbers. If they compare equal (that is, when subtracted, the result is 0), then *string* is selected for the result.

The comparison is effected using a numerical comparison with a digits setting that is either nine or, if greater, the larger of the number of digits in the mantissas of the two numbers (excluding leading insignificant zeros).

The selected result is formatted by adding zero to the selected number with a digits setting that is either nine or, if greater, the number of digits in the mantissa of the number (excluding leading insignificant zeros). Scientific notation is used, if necessary.

```
0.max(1)          ==1
v=.rexx('-1'); v.max(1)       ==1
w=.rexx('+1'); w.max(-1)      ==1
x=.rexx('1.0'); x.max(1.00)   =='1.0'
y=.rexx('1.00'); y.max(1.0)   =='1.00'
z=.rexx('123456700000'); z.max(1234567E+5)   == '123456700000'
v1=.rexx('1234567E+5'); v1.max('123456700000') == '1.234567E+11'
```

11.31 min(number)

returns the smaller of *string* and *number*, which must both be numbers. If they compare equal (that is, when subtracted, the result is 0), then *string* is selected for the result.

The comparison is effected using a numerical comparison with a digits setting that is either nine or, if greater, the larger of the number of digits in the mantissas of the two numbers (excluding leading insignificant zeros).

The selected result is formatted by adding zero to the selected number with a digits setting that is either nine or, if greater, the number of digits in the mantissa of the number (excluding leading insignificant zeros). Scientific notation is used, if necessary.

```
0.min(1)          ==0
v=.rexx('-1'); v.min(1)       =='-1'
w=.rexx('+1'); w.min(-1)      =='-1'
x=.rexx('1.0'); x.min(1.00)   =='1.0'
```

```
y=.rexx('1.00'); y.min(1.0)    =='1.00'
z=.rexx('123456700000'); z.min(1234567E+5)    == '123456700000'
v1=.rexx('1234567E+5'); v1.min('123456700000') == '1.234567E+11'
```

11.32 overlay(new [,n [,length [,pad]]])

overlays the string *new*, padded or truncated to length *length*, onto a copy of the target *string* starting at the *nth* character; the string with any overlays is returned. Overlays may extend beyond the end of the original *string*. If *length* is specified it must be a non-negative whole number. If *n* is greater than the length of the target string, padding is added before the *new* string also. The default *pad* character is a blank, and the default value for *n* is 1. *n* must be greater than 0. The default value for *length* is the length of *new*. Invalid starts, lengths, or pads raise `INVALID_ARGUMENTS`.

```
v=.rexx('abcdef'); v.overlay(' ',3)      == 'ab def'
v=.rexx('abcdef'); v.overlay('.',3,2)    == 'ab. ef'
w=.rexx('abcd'); w.overlay('qq')         == 'qqcd'
w=.rexx('abcd'); w.overlay('qq',4)       == 'abcqq'
x=.rexx('abc'); x.overlay('123',5,6,'+') == 'abc+123+++'
```

The receiver is always the target and *new* is the first method argument. See the typed `overlay` contract and separate Classic `rxfnsc/overlay` contract.

11.33 pos(needle [,start])

returns the position of the string *needle*, in *string* (the ”haystack”), searching from left to right. If the string *needle* is not found, or is the null string, 0 is returned. By default the search starts at the first character of *string* (that is, *start* has the value 1). This may be overridden by specifying *start* (which must be a positive whole number), the point at which to start the search; if *start* is greater than the length of *string* then 0 is returned. Zero or negative *start* values raise `INVALID_ARGUMENTS`.
Examples:

```
v=.rexx('Saturday'); v.pos('day')     == 6
w=.rexx('abc def ghi'); w.pos('x')    == 0
w=.rexx('abc def ghi'); w.pos(' ')    == 4
w=.rexx('abc def ghi'); w.pos(' ',5) == 8
```

See the typed `pos` contract.

11.34 reverse()

returns a copy of *string*, swapped end for end.

```
v=.rexx('ABc.'); v.reverse        == '.cBA'
w=.rexx('XYZ '); w.reverse        == ' ZYX'
x=.rexx('Tranquility'); x.reverse == 'ytiliuqnarT'
```

11.35 right(length [,pad])

returns a string of length *length* containing the right-most *length* characters of *string* - that is, padded with *pad* characters (or truncated) on the left as needed. The default *pad* character is a blank. *length* must be a non-negative whole number. Invalid lengths or pads raise `INVALID_ARGUMENTS`.

```
v=.rexx('abc  d'); v.right(8)  == '  abc  d'
w=.rexx('abc def'); w.right(5) == 'c def'
x=.rexx('12'); x.right(5,'0')  == '00012'
```

See the typed `right` contract.

11.36 sequence(final)

returns a string of all characters, in ascending order of encoding, between and including the character in *string* and the character in *final*. *string* and *final* must be single characters; if *string* is greater than *final*, an error is reported.

```
v=.rexx('a'); v.sequence('f')           == 'abcdef'
w=.rexx('\\0'); w.sequence('\\x03')       == '\\x00\\x01\\x02\\x03'
x=.rexx('\\ufffe'); x.sequence('\\uffff') == '\\ufffe\\uffff'
```

11.37 xrange(final)

returns the inclusive legacy byte-domain range from the single character in *string* through the single character in *final*. Endpoints are limited to U+0000 through U+00FF, and a descending range wraps at U+00FF. Invalid endpoints signal `INVALID_ARGUMENTS`. Use `sequence` for non-wrapping Unicode ranges.

```
v=.rexx('a'); v.xrange('f').toString() == 'abcdef'
```

The native contract is documented in `xrange.md`. Classic Level C XRANGE uses configuration-coded characters and is documented separately in `rxfnsc/xrange.md`.

11.38 sign()

returns a number that indicates the sign of *string*, which must be a number. *string* is first formatted, just as though the operation "**string+0**" had been carried out with sufficient digits to avoid rounding. If the number then starts with '-' then **'-1'** is returned; if it is **'0'** then **'0'** is returned; and otherwise **'1'** is returned.

```
v=.rexx('12.3'); v.sign    ==  1
w=.rexx('0.0'); w.sign     ==  0
x=.rexx(' -0.307'); x.sign == -1
```

The method returns a `.rexx` value containing `-1`, `0`, or `1`, preserving fluent class behavior. It accepts the same Classic sign-whitespace form as `abs()` and raises `CONVERSION_ERROR` for invalid numeric text. See the typed `sign` contract and separate Classic `rxfnsc/sign` contract.

11.39 space([n [,pad]])

returns a copy of *string* with the blank-delimited words in *string* formatted with *n* (and only *n*) *pad* characters between each word. *n* must be a non-negative whole number. If *n* is 0, all blanks are removed. Leading and trailing blanks are always removed. The default for *n* is 1, and the default *pad* character is a blank.

```
v=.rexx('abc  def  '); v.space        == 'abc def'
w=.rexx('  abc def '); w.space(3)     == 'abc   def'
v=.rexx('abc  def  '); v.space(1)     == 'abc def'
v=.rexx('abc  def  '); v.space(0)     == 'abcdef'
v=.rexx('abc  def  '); v.space(2,'+') == 'abc++def'
```

11.40 strip([option [,char]]])

returns a copy of *string* with Leading, Trailing, or Both leading and trailing characters removed, when the first character of *option* is L, T, or B respectively (these may be given in either uppercase or lowercase). The default is B. When *char* is omitted, the complete Unicode whitespace set is removed. When supplied, *char* must be exactly one character and only that character is removed; an explicit blank therefore differs from omission.

```
v=.rexx('  ab c  '); v.strip        == 'ab c'
v=.rexx('  ab c  '); v.strip('L')   == 'ab c  '
v=.rexx('  ab c  '); v.strip('t')   == '  ab c'
w=.rexx('12.70000'); w.strip('t','0') == '12.7'
x=.rexx('0012.700'); x.strip('b','0') == '12.7'
y=.rexx('  ab'); y.strip() == 'ab'
```

See the typed `strip` contract and the separate Classic `rxfnsc/strip` contract.

11.41 substr(n [,length [,pad]])

returns the sub-string of *string* that begins at the *nth* character, and is of length *length*, padded with *pad* characters if necessary. *n* must be a positive whole number, and *length* must be a non-negative whole number. If *n* is greater than *string*.**length**, then only pad characters can be returned. If *length* is omitted it defaults to be the rest of the string (or 0 if *n* is greater than the length of the string). The default *pad* character is a blank. Invalid starts, supplied lengths, or pads raise `INVALID_ARGUMENTS`.

```
v=.rexx('abc'); v.substr(2)       == 'bc'
v=.rexx('abc'); v.substr(2,4)     == 'bc  '
v=.rexx('abc'); v.substr(5,4)     == '    '
v=.rexx('abc'); v.substr(2,6,'.') == 'bc....'
v=.rexx('abc'); v.substr(5,6,'.') == '......'
```

Note: In some situations the positional (numeric) patterns of parsing templates are more convenient for selecting sub-strings, especially if more than one sub-string is to be extracted from a string.

See the typed `substr` contract and the separate Classic `rxfnsc/substr` contract.

11.42 subword(n [,length])

returns the sub-string of *string* that starts at the *nth* word, and is up to *length* blank-delimited words long. *n* must be a positive whole number; if greater than the number of words in the string then the null string is returned. *length* must be a non-negative whole number. If *length* is omitted it defaults to be the remaining words in the string. The returned string will never have leading or trailing blanks, but will include all blanks between the selected words.

```
v=.rexx('Now is the  time'); v.subword(2,2) == 'is the'
v=.rexx('Now is the  time'); v.subword(3)   == 'the  time'
v=.rexx('Now is the  time'); v.subword(5)   == ''
```

11.43 translate(tableo, tablei [,pad])

returns a copy of *string* with each character in *string* either unchanged or translated to another character.

The **translate** method acts by searching the input translate table, *tablei*, for each character in *string*. If the character is found in *tablei* (the first, leftmost, occurrence being used if there are duplicates) then the corresponding character in the same position in the output translate table, *tableo*, is used in the result string; otherwise the original character found in *string* is used. The result string is always the same length as *string*.

The translate tables may be of any length, including the null string. The output table, *tableo*, is padded with *pad* or truncated on the right as necessary to be the same length as *tablei*. The default *pad* is a blank.

```
v=.rexx('abbc'); v.translate('&','b')            == 'a&&c'
w=.rexx('abcdef'); w.translate('12','ec')        == 'ab2d1f'
w=.rexx('abcdef'); w.translate('12','abcd','.') == '12..ef'
x=.rexx('4123'); x.translate('abcd','1234')      == 'dabc'
x=.rexx('4123'); x.translate('hods','1234')      == 'shod'
```

Note: The last two examples show how the **translate** method may be used to move around the characters in a string. In these examples, any 4-character string could be specified as the first argument and its last character would be moved to the beginning of the

```
v=.rexx('gh.ef.abcd'); v.translate(19970827,'abcdefgh')
```

(which returns "**27.08.1997**") shows how a string (in this case perhaps a date) might be re-formatted and merged with other characters using the **translate** method.

11.44 trunc([n])

returns the integer part of *string*, which must be a number, with *n* decimal places (digits after the decimal point). *n* must be a non-negative whole number, and defaults to zero.

The number *string* is formatted by adding zero with a digits setting that is either nine or, if greater, the number of digits in the mantissa of the number (excluding leading insignificant zeros). It is then truncated to *n* decimal places (or trailing zeros are added if needed to make up the specified length). If *n* is 0 (the default) then an integer with no decimal point is returned. The result will never be in exponential form.

```
v=.rexx('12.3'); v.trunc         == 12
w=.rexx('127.09782'); w.trunc(3) == 127.097
x=.rexx('127.1'); x.trunc(3)     == 127.100
y=.rexx('127'); y.trunc(2)       == 127.00
z=.rexx('0'); z.trunc(2)         == 0.00
```

The receiver uses the same Classic numeric-text normalization as `abs()` and `sign()`; invalid text raises `CONVERSION_ERROR`, while a negative *n* raises `INVALID_ARGUMENTS`. See the typed `trunc` contract and separate Classic `rxfnsc/trunc` contract.

11.45 upper()

Returns a copy of the complete receiver with its Unicode lowercase characters converted to uppercase. The `.Rexx` method has no start or length extension.

```
v=.rexx('Fou-Baa'); v.upper().toString() == 'FOU-BAA'
w=.rexx('école'); w.upper().toString() == 'ÉCOLE'
x=.rexx(''); x.upper().toString() == ''
```

See the native `upper` contract.

11.46 verify(reference [,option [,start]])

verifies that *string* is composed only of characters from *reference*, by returning the position of the first character in *string* that is not also in *reference*. If all the characters were found in *reference*, 0 is returned. The *option* may be either **‘Nomatch’** (the default) or **'Match'**. Only the first character of *option* is significant and it may be in uppercase or in lowercase. If **'Match'** is specified, the position of the first character in *string* that **is** in *reference* is returned, or 0 is returned if none of the characters were found. The default for *start* is 1 (that is, the search starts at the first character of *string*). This can be overridden by giving a different *start* point, which must be positive. If *string* is the null string, the method returns 0, regardless of the value of the *option*. Similarly if *start* is greater than *string***.length**, 0 is returned. If *reference* is the null string, then the returned value is the same as the value used for *start*, unless **'Match'** is specified as the *option*, in which case 0 is returned.

```
v=.rexx('123'); v.verify('1234567890')          == 0
w=.rexx('1Z3'); w.verify('1234567890')          == 2
x=.rexx('AB4T'); x.verify('1234567890','M')     == 3
y=.rexx('1P3Q4'); y.verify('1234567890','N',3)  == 4
z=.rexx('ABCDE'); z.verify('','n',3)            == 3
v1=.rexx('AB3CD5'); v1.verify('1234567890','m',4) == 6
```

11.47 word(n)

returns the *nth* blank-delimited word in *string*. *n* must be positive. If there are fewer than *n* words in *string*, the null string is returned. This method is exactly equivalent to *string***.subword(***n***,1)**.

```
v=.rexx('Now is the time'); v.word(3) == 'the'
v=.rexx('Now is the time'); v.word(5) == ''
```

11.48 wordindex(n)

returns the character position of the *nth* blank-delimited word in *string*. *n* must be positive. If there are fewer than *n* words in the string, 0 is returned.

```
v=.rexx('Now is the time'); v.wordindex(3) == 8
v=.rexx('Now is the time'); v.wordindex(6) == 0
```

11.49 wordlength(n)

returns the length of the *nth* blank-delimited word in *string*. *n* must be positive. If there are fewer than *n* words in the string, 0 is returned.

```
v=.rexx('Now is the time'); v.wordlength(2)    == 2
w=.rexx('Now comes the time'); w.wordlength(2) == 5
v=.rexx('Now is the time'); v.wordlength(6)    == 0
```

11.50 wordpos(phrase [,start])

searches *string* for the first occurrence of the sequence of blank-delimited words *phrase*, and returns the word number of the first word of *phrase* in *string*. Multiple blanks between words in either *phrase* or *string* are treated as a single blank for the comparison, but otherwise the words must match exactly. Similarly, leading or trailing blanks on either string are ignored. If *phrase* is not found, or contains no words, 0 is returned. By default the search starts at the first word in *string*. This may be overridden by specifying *start* (which must be positive), the word at which to start the search.

```
v=.rexx('now is the time'); v.wordpos('the')       == 3
v=.rexx('now is the time'); v.wordpos('The')       == 0
v=.rexx('now is the time'); v.wordpos('is the')    == 2
v=.rexx('now is the time'); v.wordpos('is    the') == 2
v=.rexx('now is the time'); v.wordpos('is  time')  == 0
w=.rexx('To be or not to be'); w.wordpos('be')     == 2
w=.rexx('To be or not to be'); w.wordpos('be',3)   == 6
```

11.51 words()

returns the number of blank-delimited words in *string*.

```
v=.rexx('Now is the time'); v.words == 4
w=.rexx(' '); w.words               == 0
x=.rexx(''); x.words                == 0
```

11.52 x2b()

Converts each hexadecimal digit in the receiver to four binary digits. Letters are case-insensitive, leading zero nibbles are retained, and an empty receiver returns empty. Interior blanks are accepted only when an even number of hexadecimal digits lies to their right. Invalid text raises `INVALID_ARGUMENTS`.

The method accepts no arguments and returns a REXX string object.

```
v=.rexx('C3'); v.x2b().toString() == '11000011'
w=.rexx('7'); w.x2b().toString() == '0111'
x=.rexx('1 C1'); x.x2b().toString() == '000111000001'
y=.rexx('0001'); y.x2b().toString() == '0000000000000001'
```

The native function contract is documented in `x2b.md`. Classic Level C X2B is documented separately in `rxfnsc/x2b.md`.

11.53 x2c()

Parses hexadecimal bytes from the receiver and maps each byte to the Unicode code point U+0000 through U+00FF. An odd leading nibble is padded with zero; empty input returns empty and leading zero bytes are retained. Values above `7F` are encoded as valid UTF-8 text, not copied as raw bytes.

Interior blanks are valid only when an even number of hexadecimal digits lies to their right. Invalid text raises `INVALID_ARGUMENTS`.

```
v=.rexx('416263'); v.x2c().toString() == 'Abc'
w=.rexx('4d'); w.x2c().toString() == 'M'
x=.rexx('FF'); x.x2c().toString() == 'ÿ'
```

The native function contract is documented in `x2c.md`. Classic Level C X2C uses a configuration-coded character service and is documented separately in `rxfnsc/x2c.md`.

11.54 x2d([n])

Hexadecimal to decimal. Converts the *string* (a string of hexadecimal characters) to a decimal number, without rounding. If *string* is the null string, 0 is returned.

If *n* is not specified, *string* is taken to be an unsigned number.

```
v=.rexx('0E'); v.x2d    == 14
w=.rexx('81'); w.x2d    == 129
x=.rexx('F81'); x.x2d   == 3969
y=.rexx('FF81'); y.x2d  == 65409
z=.rexx('c6f0'); z.x2d  == 50928
```

If *n* is specified, *string* is taken as a signed number expressed in *n* hexadecimal characters. If the most significant (left-most) bit is zero then the number is positive; otherwise it is a negative number in twos-complement form. In both cases it is converted to a number which may, therefore, be negative. If *n* is 0, 0 is always returned.

If necessary, *string* is padded on the left with **'0'** characters (note, not "sign-extended"), or truncated on the left, to length *n* characters; (that is, as though *string*.**right(***n*, **'0')** had been executed.)

```
v=.rexx('81'); v.x2d(2)   == -127
v=.rexx('81'); v.x2d(4)   == 129
w=.rexx('F081'); w.x2d(4) == -3967
w=.rexx('F081'); w.x2d(3) == 129
w=.rexx('F081'); w.x2d(2) == -127
w=.rexx('F081'); w.x2d(1) == 1
x=.rexx('0031'); x.x2d(0) == 0
```

The **c2d** method can be used to convert a character to a decimal representation of its encoding.

This method is the native Level B signed-64-bit surface. Invalid hexadecimal text or a negative *n* signals `INVALID_ARGUMENTS`; a result outside the native range signals `OVERFLOW_UNDERFLOW`. Its optional argument is presence-aware, so an omitted width is unsigned while an explicit zero returns zero. See the native function contract. Classic Level C X2D is an arbitrary-width RexxValue BIF with caller `NUMERIC DIGITS` handling, documented separately in `rxfnsc/x2d.md`.

12

Concurrency classes

The `concurrency` namespace is the object layer over cREXX's provider-neutral channel instructions. It supplies bounded task pools, structured scopes, canonical values, byte endpoints and explicit binary ownership. The language chapter adds Level G task syntax on top; the classes themselves are available at Level B.

The surface is initial on `develop`. Local-thread and isolated-process providers are qualified on macOS in the current tree. Portable publication is still subject to Linux and Windows conformance.

12.1 Import and object relationships

```
options levelb
import rxfnsb
import concurrency
```

A pool and a scope have different jobs:

- a `.taskpool` is a bounded execution provider and capacity policy;
- a `.taskscope` owns child lifetime, cancellation, deadline and joining;
- a `.task` is one submitted child; and
- a `.completion` is an immutable terminal observation.

One pool may serve several scopes. A scope is consumed once and cannot be reused. Close every scope before closing its pool. Ordinary tasks cannot detach from their scope.

```
taskpool (capacity) --> taskscope (lifetime) --> task --> completion
                            |
                            +--> task --> completion
```

The common transport layer has the same ownership shape:

```
channel (provider owner) --> channelrequest --> completion
```

Raw channel handles and request tickets are deliberately private. The wrapper objects prevent an integer from being mistaken for transferable authority.

12.2 Pools and structured scopes

Create a local worker-thread pool or an isolated worker-process pool:

```
local_pool = .taskpool.local(4, 64)
process_pool = .taskpool.process(2, 16)
```

The first argument is maximum worker capacity. The second is bounded admission capacity. Provider type `1` is local execution and type `2` is process isolation. Capacity permits overlap; it does not promise a particular schedule.

Choose a scope policy and relative deadline:

```
scope = .taskscope.failfast(local_pool, 30000)
scope = .taskscope.collectall(local_pool, 30000)
```

`failfast` requests cancellation of unfinished siblings after the first unsuccessful child. `collectall` lets the other children finish. Both account for every accepted child. Timeout values are milliseconds: `-1` means no deadline or an indefinite wait, `0` is an immediate poll, and a positive value is a bounded relative wait.

The most Rexx-like way to use these objects is through Level G task syntax:

```
options levelg comments_dash
import concurrency

adderwork: class implements .taskwork
  _delta = .int

  *: factory
    arg delta = .int
    _delta = delta
    return

  run: method = .channelvalue
```

```
    arg request = .channelvalue, context = .taskcontext
    if context.cancellation_requested() then return .channelvalue.
        null_value()
    return .channelvalue.integer_value(request.as_integer() + _delta)

main: procedure = .int
  pool = .taskpool.local(1, 4)
  scope = .taskscope.collectall(pool, 60000)

  target = task .adderwork(2)
  child = scope.submit(target, .channelvalue.integer_value(40))
  outcome = child.wait(-1)

  if outcome.succeeded() then answer = outcome.value().as_integer()
  else do
    say outcome.error_code() outcome.message()
    answer = -1
  end

  call scope.finish()
  call pool.close()
  return answer <> 42

exit main()
```

This exact pattern is compiled, assembled, linked and run in `concurrency_-taskwork.crexx`. `task .adderwork(2)` is Level G syntax: the compiler, assembler and linker create and seal the target descriptor. Application code must not call `.tasktarget.binding()` with invented bytes.

The main scope operations are:

Operation	Meaning
`submit(target, request)`	submit one `.channelvalue` application request
`next(milliseconds)`	observe the next child in completion order
`join()`	return all child completions in submission order
`cancel(reason)`	request cancellation of unfinished children
`finish()`	join, close and raise `TASK_FAILURE` on child failure
`abort(reason)`	cancel, join and close during abnormal controller exit

`next()` may return an unavailable sentinel with `available() = 0`. That is a timed or nonblocking miss, not a child completion, and it never appears in `join()`.

`.taskscope.ask()` is reserved for future single-owner services and currently signals unsupported status `19`. `.taskpool.queued()` and `.running()` are also reserved and signal `19`; they do not return guessed telemetry.

12.3 Task work and task context

`.taskwork` is the advanced runnable contract below ordinary typed task calls:

```
run: method = .channelvalue
  arg request = .channelvalue, context = .taskcontext
```

A factory target such as `task .adderwork(2)` creates a worker-owned class instance. The application `request` is separate from the factory arguments. The current factory-target lowering accepts `.boolean`, `.int`, `.string`, `.binary` and direct `.channelvalue` arguments.

The context reports:

- `timeout_remaining()`: remaining milliseconds, or `-1` without a deadline;
- `cancellation_requested()`: cooperative cancellation state; and
- `trace_identity()`: stable invocation identity.

`endpoint(reference)` adapts a transferable type-4 provider reference to a worker-local `.byteendpoint`. Use it inside `.taskwork.run()` when a task must read or write an endpoint owned by its controller; ordinary controller code can use `.byteendpoint.from_reference()` directly.

`.taskarguments` and the `submit_arguments()` / `result_*()` methods are exposed compiler-lowering bridges. They are documented so generated code is auditable, but ordinary programs should use typed task calls or `submit(target, request)`.

12.4 Completion states

Always test `available()` before treating an observation as terminal, then use `succeeded()`, `value()`, `error_code()`, `message()` and `details()` rather than guessing from a message string.

State	Meaning
`0`	unavailable observation sentinel
`1`	succeeded
`2`	failed
`3`	cancelled
`4`	deadline exceeded
`5`	rejected or target not found
`6`	endpoint closed or provider shutdown
`7`	transport lost before execution
`8`	outcome unknown after transport loss
`9`	isolated task killed

State is provider-neutral terminal classification. `error_code()` carries the more specific operation/provider status. `value()` is valid only when `has_value()` is true.

12.5 Canonical channel values

`.channelvalue` is the only ordinary value document accepted at a provider boundary. It can represent null, boolean, integer, float, decimal, string, binary, arrays and named versioned records. The receiver owns a validated new value; it does not receive a pointer to the sender's object.

```
names = .string[]
values = .channelvalue[]
names[1] = "name"
values[1] = .channelvalue.string_value("Ada")
names[2] = "score"
values[2] = .channelvalue.integer_value(42)

record = .channelvalue.record_value("example.result", 1, names, values)
say record.schema() record.version()
say record.field("score").as_integer()
```

Record field names must be unique. Construction writes canonical byte-name order, so equivalent records encode identically. Accessors validate the value kind: for example `as_integer()` does not coerce a string.

`.channelcodec` is the explicit application codec interface:

```
schema: method = .string
version: method = .int
encode: method = .channelvalue
  arg item = .object
```

```
decode: method = .object
  arg item = .channelvalue
```

There is no ambient runtime codec registry. Level G transferable concrete classes instead use their statically resolved `to_channel()` and `from_channel()` members.

12.6 Channels and byte endpoints

Most programs use pools, scopes and `.byteendpoint` rather than opening a raw `.channel`. The channel object directly exposes the provider lifecycle:

```
channel = .channel.open(provider_type, required_capabilities,
    configuration)
request = channel.start(envelope, admission_wait)
outcome = request.wait(completion_wait)
if outcome.available() then call request.release()
call channel.close(1)  /* drain */
```

Close mode `1` drains accepted work; mode `2` requests cancellation. A channel owns its requests, so cancellation with a request from another channel is an error.

Completion observation and request lifetime are separate. `channel.wait()` delivers each completion once; `request.wait()` and `request.completion()` can return that saved outcome repeatedly. Neither observation, cancellation nor dropping a request variable releases the request. For sustained use, call `request.release()` after obtaining an available terminal completion and finishing request access. Close the channel when its own lifetime ends.

Release applies to successful, failed, cancelled and deadline outcomes. It raises `CHANNEL_ERROR` with status `10` if the request is pending or its completion has not been successfully observed; a polling miss is not sufficient. Release does not cancel or consume anything in that case. To abandon pending work, cancel it, wait for an available terminal completion, then release it, or close the channel. Release waits for private request cleanup, including native thread joins. If cleanup fails, the request remains owned and release can be retried.

After successful release, all copies of that request are stale: wait, completion, terminal, cancel and repeated release raise `CHANNEL_ERROR` with status `13`. Its diagnostic identity remains readable. Saved `.completion` values remain usable after release and channel close. Caller-retained completion values still occupy their own memory; release removes the channel's request and cache ownership. Ordinary VM value buffers can retain reusable capacity while wrapper storage remains alive, so release and close do not promise an immediate drop in process memory. Provider request payloads and private thread resources are destroyed before successful release returns. This does not add a `.task.release()` or allow structured children to detach. Task scopes retain their join results for task-result access after close.

The runtime permits at most 65,535 unreleased request tickets per execution, shared across channels and including completed requests. Released slots are reused; generation wrap retires slots to protect stale capabilities, so the authority namespace is finite. Provider admission, allocation and thread limits are separate. Raising status `8` does not identify which limit was reached.

The new release instruction has a separate RXBIN feature bit. Update the runtime and class library together and rebuild consumers of `release()`. Older runtimes reject images that require it. Existing callers keep their earlier ownership behavior until they adopt release or close their channels.

Provider type `4` supplies reusable bounded byte endpoints. This complete Level B example writes and reads through one duplex endpoint:

```
options levelb
import rxfnsb
import concurrency

endpoint = .byteendpoint.memory(3, 32)

write_request = endpoint.start_write("hello" as .binary, -1)
write_result = write_request.wait(-1)
call write_request.release()
if write_result.succeeded() = 0 then exit 1

read_request = endpoint.start_read(5, -1)
read_result = read_request.wait(-1)
call read_request.release()
if read_result.succeeded() = 0 then exit 2
if read_result.value().as_binary() as .string <> "hello" then exit 3

call endpoint.half_close(2)
call endpoint.close()
exit 0
```

Endpoint direction is `1` readable, `2` writable or `3` duplex. Memory capacity is a real backpressure bound. A transferable provider reference can be adapted in another execution with `from_reference()` or, for an exact encoded 92-byte reference, `from_encoded_reference()`. Close every adapter when finished.

`half_close()` and provider-reference export release their temporary requests internally. Explicit `start_read()` and `start_write()` return caller-owned requests and require caller release. A half-close still needs a new request, so it can fail at ticket exhaustion. Arrange cleanup so that a failed preliminary operation does not prevent whole-channel close, which needs no new ticket.

12.7 Transfer buffers

`.transferbuffer` makes binary ownership transitions visible:

```
buffer = .transferbuffer.copy_from("answer=41" as .binary)
call buffer.write(7, "42" as .binary)
snapshot = buffer.seal()
say snapshot.as_binary() as .string
```

The states are:

- **mutable**: read, write, `move_value()` or `seal()`;
- **moved**: the immutable value owns the transferred bytes and the source may no longer be read or changed; and
- **sealed**: the buffer retains read-only access and repeated `seal()` calls return the same stable immutable snapshot.

A transfer-buffer seal is not encryption, authentication or a checksum. It only prevents later writes through that buffer. It is unrelated to the sealed task-binding integrity descriptor.

12.8 Reserved surfaces

`.serviceref` records the intended logical identity shape for a future single-owner service. There is no concrete public service, no working `.taskscope.ask()` path and no typed service proxy today. Provider type `3` (open host) and the public provider-plugin ABI are also reserved.

For task-call syntax and more approachable examples, see Concurrent programming. For the formal Level G rules, see the Concurrency language reference.

13 Concurrent HTTP client and server

The `rxfnsg` namespace supplies a bounded HTTP/1.1 client and a clear-text buffered server for Level G programs. The client combines structured task scopes, single-owner reusable connections, safe transferable headers and endpoint-backed streaming. The server keeps sockets on one controller and gives complete request values to ordinary task classes. HTTP is a REXX library built on concurrency; it is not a special RXAS instruction or channel provider type.

The client is initial on `develop` and currently macOS-qualified. Linux and Windows publication still requires the portable concurrency conformance matrix.

13.1 One public surface, one private backend

User programs import `rxfnsg` and use these public Level G types:

Purpose	Public types	Execution model
client	`.httpclient`, `.httpheaders`, `.httppolicy`, `.httpbody`, `.httpresponse`	task calls over bounded single-owner connections
server	`.httpserver`, `.httprequest`, `.httpservice`, `.httpresponse`	controller-owned sockets and sealed handler tasks
LLM providers	`.ollama`, `.openai`, `.anthropic`, `.gemini`	the same Level G client and owner pool

All three use `_rxhttpcore`, one private binary-oriented Level B framing, parsing and codec backend. It is deliberately not a second public HTTP client: Level B supplies the socket and binary machinery, while Level G owns HTTP policy, resource management and typed values. The former public `rxhttp` convenience client has been removed from the pre-release surface.

13.2 Import and first parallel requests

```
options levelg comments_dash
import rxfnsb
import concurrency
import rxfnsg
```

Create headers, a per-origin client and a task scope, then use ordinary method syntax inside `DO PARALLEL`:

```
headers = .httpheaders.json()
call headers.add("Authorization", "Bearer " || token)
call headers.add("Idempotency-Key", request_id)

client = .httpclient.pooled("https://api.example", 4, 16, 1048576)
request_pool = .taskpool.local(2, 8)
request_scope = .taskscope.failfast(request_pool, 60000)

do parallel using request_scope
  generation = client.post("/v1/chat/completions", generation_json,
      headers)
  embedding = client.post("/v1/embeddings", embedding_json, headers)
end

if generation.succeeded() then say generation.body()
else say generation.transport_status() generation.error()

call request_pool.close()
call client.close()
```

`post()` is a task method. The task pool bounds Rexx task executions while the HTTP client independently bounds reusable connections and queued admissions. The complete loopback generation/embedding fixture is `ts_http_crexx_rag.crexx`. It verifies authorization and idempotency headers, request shapes, parallel admission and bounded gzip/DEFLATE response decoding.

For a buffered GET use `client.get(path, headers)`. For another HTTP method or a binary request body use `client.request(verb, path, body_bytes, headers)`.

All three forms return the same `.httpresponse`; use `body()` for text and `body_bytes()` for exact bytes.

13.3 Client factory and lifecycle

```
client = .httpclient.pooled(origin, connections, admission,
    maximum_response)
client = .httpclient.pooled(origin, connections, admission,
    maximum_response, policy)
```

Argument	Contract
`origin`	`http://` or `https://` origin; IPv6 literals use brackets
`connections`	`1..32` single-owner reusable connections
`admission`	`1..256` queued fixed-size request descriptors
`maximum_response`	`1..8388608` decoded bytes for buffered responses
`policy`	optional `.httppolicy`, default `.httppolicy.safe()`

Request paths must begin with `/`, occupy at most 8192 UTF-8 bytes and contain no CR or LF. Each configured connection has one long-lived `.taskwork` owner; its socket integer never crosses an execution boundary.

Only the controller-owned client should perform final `close()`. That closes admission, joins the connection owners and releases their pool. A client proxy reconstructed for a task closes only its local endpoint view. Close the request task pool separately, after every scope using it has closed.

`from_channel()` and `to_channel()` appear in the generated API because they are the exact compiler transfer contract. Ordinary callers use `pooled()`, `post()`, `post_stream()` and `close()`.

The same rule applies to `from_channel()` / `to_channel()` on headers, policy, body and response objects. `httpresponse.new()`, `from_wire()`, `from_stream_wire()` and `attach_stream()` are library construction/framing bridges. They remain documented because they are callable in the current pre-release class surface, but normal request code should consume responses returned by the client.

13.4 Safe headers

Construct a validated transferable snapshot with:

```
headers = .httpheaders.empty()
headers = .httpheaders.json()
headers = .httpheaders.media_type("application/cbor")

call headers.add("Authorization", "Bearer " || token)
call headers.add("Idempotency-Key", request_id)
```

Header names must be HTTP tokens. Values and `Content-Type` cannot contain CR or LF. `Host`, `Content-Length`, `Connection`, `Transfer-Encoding` and `Accept-Encoding` are library-owned. The absolute snapshot ceiling is 256 custom fields and 262144 bytes. The active policy may impose smaller limits.

Useful accessors are `content_type()`, `count()`, `bytes()` and `header(name)`. `wire()` is the validated CR/LF-terminated internal wire block. `to_channel()` / `from_channel()` are transfer-contract members; a reconstructed snapshot is fully validated again, so forged canonical data cannot bypass the header rules.

13.5 Policy, limits and deadlines

```
policy = .httppolicy.safe()
call policy.set_timeouts(10000, 10000, 10000, 30000, 30000, 60000)
call policy.set_limits(64, 65536, 8388608)
call policy.set_replay(0, 0)
```

The timeout order is DNS, connect, TLS, request, response and total. The limit order is custom-header count, header bytes and buffered request bytes. Safe defaults are:

Policy	Default	Valid range
DNS, connect, TLS	10000 ms each	`1..600000`
request, response	30000 ms each	`1..600000`
total observation	60000 ms	`1..1800000`
custom header fields	64	`1..256`
header bytes	65536	`256..262144`
buffered request bytes	8388608	`1..8388608`
redirects, retries	0	`0..8` each

Validation happens before mutation, so a rejected setter does not leave a partially changed policy. The present socket substrate performs name resolution, connect and TLS handshake as one owner-local operation; the client uses the sum of those three phase budgets for that atomic call. Request and response socket work use their individual budgets. The containing task scope should still express the caller's real end-to-end deadline.

13.6 Buffered responses and compression

`post(path, body, headers)` accepts a buffered `.string` body and returns an independent `.httpresponse`. It sends `Accept-Encoding: gzip, deflate` and performs bounded decoding for:

- gzip;
- zlib-wrapped DEFLATE; and
- raw DEFLATE.

The decoded result may not exceed `maximum_response`. Malformed streams, unsupported content coding and decoded-size overflow are explicit transport failures; the client does not return partial decoded data as success.

Inspect responses as follows:

```
if response.transport_status() <> 0 then do
  say "transport/protocol failure:" response.transport_status()
      response.error()
end
else do
  say "HTTP status:" response.status()
  say "Content-Type:" response.header("Content-Type")
  if response.succeeded() then say response.body()
end
```

`succeeded()` means transport status zero and HTTP status in `200..299`. `attempts()` and `redirects()` report network history. `ambiguous_outcome()` remains true when bytes may have reached the peer without a definitive response, even if an explicitly permitted retry later succeeds.

13.7 Redirects, retries and ambiguous POST outcomes

Automatic replay is disabled by default. `set_replay()` sets only bounded maximums; it cannot make an unsafe POST idempotent.

- a connect failure before any bytes are sent may be retried within policy;
- a retry after sending requires a non-empty `Idempotency-Key`;
- retryable statuses are 408, 425, 429, 500, 502, 503 and 504, again requiring the key;
- only same-origin 307 and 308 retain POST and may be followed;
- 301, 302 and 303 POST rewriting is deliberately disabled; and
- cross-origin redirects are returned and never followed.

Callers must treat `ambiguous_outcome()` as delivery-history information. A successful later response does not prove the earlier attempt was unseen.

13.8 Streaming request and response bodies

Use `.httpbody.fixed(content_length, capacity, response_capacity)` when the producer knows the exact length. Use `.httpbody.chunked(capacity, response_capacity)` otherwise. Request capacity is `1..16777216`; response capacity is `256..16777216`.

The producer must run concurrently with `post_stream()` or a small endpoint can correctly backpressure both sides. This fixed-length example is adapted from the executable streaming fixture:

```
produce: task = .int
  arg body = .httpbody

  if body.write("hello " as .binary, 5000) <> 6 then return 1
  if body.write("world" as .binary, 5000) <> 5 then return 2
  call body.finish()
  return 0

body = .httpbody.fixed(11, 8, 512)
scope = .taskscope.failfast(request_pool, 60000)

do parallel using scope
  produced = produce(body)
  response = client.post_stream("/upload", body, headers, 512)
end
```

```
if produced <> 0 then say "producer failed" produced
```

The `response_capacity` argument to `post_stream()` must exactly equal `body.response_capacity()`. `write(bytes, milliseconds)` returns accepted bytes or `-1` after cancellation, deadline or failure. `finish()` half-closes the request write side and publishes EOF. `cancel(reason)` cancels both paired endpoints; `close()` releases both adapters.

A successful streaming response contains metadata plus a byte-endpoint reference. Drain it explicitly:

```
drain_response: procedure = .binary
  arg response = .httpresponse

  endpoint = .byteendpoint.from_encoded_reference(response.
      body_reference())
  result = ''x as .binary

  do forever
    request = endpoint.start_read(4096, 5000)
    outcome = request.wait(5000)
    if outcome.available() = 0 | outcome.succeeded() = 0 then do
      call endpoint.close()
      return ''x as .binary
    end

    chunk = outcome.value().as_binary()
    if binlength(chunk) > 0 then result = binconcat(result, chunk)
    if outcome.details().field("eof").as_boolean() then leave
  end

  call endpoint.close()
  return result
```

Call `streaming()` before choosing `body()` or `body_reference()`; using the wrong accessor signals an error. Streaming responses currently require identity content encoding. Buffered responses provide the bounded compression support described above. The complete fixed/chunked request and streamed response proof is `ts_http_streaming.crexx`.

13.9 TLS and verification

`https://` origins use `socketconnecttls(sock, host, port)`. Supported builds verify the certificate chain and hostname using the configured platform backend: Apple system TLS, OpenSSL on supported Unix-like systems, or SChannel on Windows. A build without TLS support fails the request; it does not downgrade to plaintext.

The live TLS fixture is environment-gated because it depends on external network and trust state. The ordinary loopback suite remains deterministic and covers pool ownership, policy, redirects, retries, ambiguity, streaming, compression and the cREXX-RAG request shapes.

13.10 A REXX task class as an HTTP service

An HTTP handler is a normal user class implementing `.httpservice` and `.taskwork`. Write the typed `handle(request, context)` method for application logic. The explicit `run` method is the bridge to the standard `.taskwork` ABI; `.httpservice.dispatch` performs the checked request/response conversion.

```
options levelg
import rxfnsb
import concurrency
import rxfnsg

health_service: class implements .httpservice .taskwork
  *: factory
    return

  handle: method = .httpresponse
    arg request = .httprequest, context = .taskcontext
    if request.verb() = "GET" & request.target() = "/health" then do
      return .httpresponse.text(200, "ok")
    end
    return .httpresponse.text(404, "not found")

  run: method = .channelvalue
    arg encoded = .channelvalue, context = .taskcontext
    service = self as .httpservice
    return service.dispatch(encoded, context)

main: procedure = .int
  server = .httpserver.local("127.0.0.1", 8080)
```

```
  served = server.serve(task .health_service(), 1)
  call server.close()
  return 0

exit main()
```

This complete example deliberately serves one request so that normal cleanup is visible. Omitting the second `serve` argument, or passing `-1`, keeps the controller loop serving until its host is stopped. Detached/background server lifecycle is not part of the current surface.

The `task .health_service()` expression is a compiler/linker-sealed factory target. Each accepted request constructs task work according to that target; the pool is a bounded executor, not a user-object factory. Factory arguments may be added in the usual class syntax, subject to normal transferable task argument rules.

13.11 Server request and response values

`.httprequest` gives a handler these read-only views:

Method	Result
`verb()`	validated uppercase request method
`target()`	origin-form target, including any query
`header(name)`	case-insensitive header lookup
`headers()`	raw validated header field block
`body()` / `body_bytes()`	buffered body as text / exact bytes
`remote()`	numeric peer endpoint text

`inbound()`, `routed()`, `from_channel()`, `to_channel()`, `routed_reply()` and `reply_-reference()` appear in the generated pre-release API because they are the exact controller/task transfer bridge. Normal handlers receive a request from `dispatch`; they do not construct routed requests or use the private reply capability themselves.

Return `.httpresponse.text(status, text)` for text or `.httpresponse.binary(status, bytes)` for exact bytes. Response headers may be supplied with the optional `.httpheaders` argument. The controller validates the returned status, headers and body against server limits before any socket write. A raised handler or an unusable response becomes a bounded HTTP 500; a handler deadline becomes 504.

13.12 Server factory, limits and lifecycle

```
server = .httpserver.local(host, port, workers, admission, connections,
     maximum_header_bytes, maximum_headers, maximum_body_bytes,
    handler_timeout, request_timeout)
```

Defaults are `4` workers, `64` admitted tasks, `64` accepted connections, 65536 header bytes, `100` headers, 8388608 body bytes and 30000 ms for both handler and request-read deadlines. Port `0` asks the operating system for an ephemeral port; `port()` returns the selected value.

`serve(target, maximum_requests)` owns the controller loop and returns the number completed by that call. Only complete requests cross to workers. The controller retains the listener and every accepted socket, permits one in-flight request per connection and uses bounded nonblocking scans with a short idle wait. `served()`, `scans()` and `idle_scans()` are diagnostic counters, not throughput guarantees. Call `close()` after a bounded `serve` returns; it closes sockets and joins the private handler scope and pool.

The initial server accepts buffered clear-text HTTP/1.1 requests, with HTTP/1.0 compatibility, canonical `Content-Length`, keep-alive and pipelining boundaries. It rejects request `Transfer-Encoding` with 501. Server TLS, HTTP/2, WebSockets, detached/background lifecycle, request streaming and streaming handler responses are outside the present contract.

The executable service, parallel-client, binary-body, malformed-message, deadline and failure examples are `ts_http_server.crexx` and `ts_http_server_failures.crexx`.

14 Unicode Text Services

rxunicode is cREXX's explicit Unicode 17.0.0 library. It normalizes text, performs full default case conversion and case folding, segments default extended grapheme clusters, and converts between .string text and .binary encoded bytes.

```
options levelg
import rxfnsb
import rxunicode

text = rxunicode..decode(readbinary("input.txt"), "Windows-1252")
text = rxunicode..toNFC(text)
text = rxunicode..toUppercase(text)
bytes = rxunicode..encode(text, "UTF-8")
call writebinary "output.txt", bytes
```

Importing rxunicode does not change ordinary string equality, codepoint indexing, Level B upper or lower, file I/O, or .binary semantics. Every potentially expensive or information-changing operation remains visible at the call site.

14.1 Text, bytes, and graphemes

When choosing a representation or indexing unit, CREXX keeps bytes, Unicode codepoints, and user-perceived characters distinct:

Concern	CREXX surface
Arbitrary or encoded bytes	.binary, readbinary, and writebinary
Valid UTF-8 text and scalar positions	.string and ordinary Level B string operations

Concern	CREXX surface
User-perceived character boundaries	explicit `rxunicode` grapheme operations

Ordinary `.string` length, position, substring, and reverse operations count Unicode codepoints. Use the grapheme family when the application means a user-perceived character. Use `decode` and `encode` when crossing a byte/text boundary; a cast alone does not express a legacy encoding.

`rxunicode..version()` returns the Unicode data version used by every versioned algorithm in the module, currently `17.0.0`.

14.2 API at a glance

The public API has two forms: stateless namespace procedures for one-off work, and an indexed grapheme snapshot for repeated boundary access. Whole-file byte I/O is supplied separately by Level B.

14.2.1 Namespace procedures

Most `rxunicode` services are namespace procedures. Their static types are:

```
version() = .string

toNFD(text = .string) = .string
toNFC(text = .string) = .string
toNFKD(text = .string) = .string
toNFKC(text = .string) = .string
isNFD(text = .string) = .boolean
isNFC(text = .string) = .boolean
isNFKD(text = .string) = .boolean
isNFKC(text = .string) = .boolean

toUppercase(text = .string) = .string
toLowercase(text = .string) = .string
toCasefold(text = .string) = .string
toSimpleCasefold(text = .string) = .string
toTurkicCasefold(text = .string) = .string
toTurkicSimpleCasefold(text = .string) = .string

graphemeCount(text = .string) = .int
graphemeSubstr(text = .string, start = .int
               [, length = .int [, pad = .string]]) = .string
```

```
graphemePos(needle = .string, haystack = .string
            [, start = .int]) = .int
graphemeReverse(text = .string) = .string

encode(text = .string [, encoding = .string
       [, replacement = .binary]]) = .binary
decode(data = .binary [, encoding = .string
       [, replacement = .string]]) = .string
isDecodable(data = .binary [, encoding = .string]) = .boolean
isEncodingSupported(encoding = .string) = .boolean
```

The codec `encoding` argument defaults to `UTF-8`. The defaults and error rules for the other optional arguments are described with the relevant operation families below.

14.2.2 Indexed grapheme classes

Repeated grapheme indexing is the one public class-based family. Calling `.rxunicode..graphemes(` captures an immutable text snapshot and prepares its boundary index once:

```
.rxunicode..graphemes(text = .string) = .graphemes
  text() = .string
  count() = .int
  at(position = .int) = .string
  substr(start = .int [, length = .int [, pad = .string]]) = .string
  pos(needle = .string [, start = .int]) = .int
  reverse() = .string
  codepointStart(position = .int) = .int
  codepointLength(position = .int) = .int
  iterator() = .graphemeiterator
  version() = .string
  profile() = .string

.graphemeiterator
  hasNext() = .int
  next() = .string
  index() = .int
  reset() = .void
  codepointStart() = .int
  codepointLength() = .int
```

Applications obtain `.graphemeiterator` from the snapshot's `iterator()` method. The iterator's packed-boundary constructor arguments are internal and are not a supported application entry point.

14.2.3 Related Level B file procedures

Whole-file byte I/O belongs to Level B rather than `rxunicode`. The related procedures are `readbinary(path = .string) = .binary` and `writebinary(path = .string, data = .binary) = .int`. The codec section below shows how to compose them with `decode` and `encode`.

14.3 Normalization

Normalization makes selected canonically or compatibly equivalent scalar sequences use a defined Unicode form:

Transform	Predicate	Meaning
`toNFD(text)`	`isNFD(text)`	Canonical decomposition
`toNFC(text)`	`isNFC(text)`	Canonical decomposition followed by composition
`toNFKD(text)`	`isNFKD(text)`	Compatibility decomposition
`toNFKC(text)`	`isNFKC(text)`	Compatibility decomposition followed by composition

Each transform accepts and returns `.string`. Each predicate accepts `.string` and returns `.boolean`.

```
decomposed = "65cc81"x        /* e plus combining acute */
composed = rxunicode..toNFC(decomposed)

if rxunicode..isNFC(composed) then say "NFC"
```

NFD and NFC preserve canonical meaning but can change codepoint count and byte representation. NFKD and NFKC may additionally remove compatibility distinctions such as presentation forms, width variants, and some formatting differences. Select compatibility normalization only when that loss is part of the application contract.

No assignment, comparison, concatenation, codec, case operation, grapheme operation, or file operation normalizes implicitly. There is no public normalizer object: direct procedures keep ownership and lifecycle simple while the runtime may use trustworthy string certificates internally.

14.4 Full default case conversion

`toUppercase(text)` and `toLowercase(text)` apply Unicode's locale-neutral full default mappings and return `.string`. They can expand their input; for example, default uppercase maps German ß to `SS`. Default lowercase includes Unicode's context-sensitive Greek final-sigma rule.

```
say rxunicode..toUppercase("Straße")
say rxunicode..toLowercase("□□")
```

The procedures do not normalize and do not select Turkish, Azerbaijani, Lithuanian, or another locale. CREXX deliberately does not expose titlecase or a locale object in this baseline. Level B `upper` and `lower` retain their existing simple runtime contracts; use the `rxunicode` names when the complete Unicode default mapping is required.

14.5 Case folding

Case folding produces a stable form for caseless matching. It is not presentation-oriented uppercasing or lowercasing.

Procedure	Result
`toCasefold(text)`	Default full folding using Unicode `C + F` mappings
`toSimpleCasefold(text)`	Default simple, non-expanding folding using `C + S` mappings
`toTurkicCasefold(text)`	Full folding with the Unicode Turkic `I` mappings
`toTurkicSimpleCasefold(text)`	Simple folding with the Unicode Turkic `I` mappings

Full default folding is the ordinary choice for locale-independent caseless keys and comparisons. It may expand: ß folds to `ss`. Simple folding maps no more than one output scalar for each input scalar but is not a substitute for full folding when expansion is semantically required.

```
key1 = rxunicode..toCasefold(name1)
key2 = rxunicode..toCasefold(name2)
if key1 == key2 then say "same caseless key"
```

The Turkic procedures explicitly select the special dotted and dotless I mappings used for Turkish and Azerbaijani. They do not change a process, task, or object locale.

Folding performs no normalization. Canonical caseless matching and `NFKC_Casefold` are separate contracts; an application that needs either must define its normalization/folding sequence. Expansion also means result indexes cannot be mapped back to source indexes without an application-owned map.

The four direct procedures are the complete case-folding API. Case folding retains no useful state between independent strings, so the library does not add a reusable case-folder object around the same operation.

14.6 Default extended grapheme clusters

The grapheme family implements Unicode Standard Annex #29 revision 47 profile `UAX29-C1-1`: default extended grapheme clusters without tailoring. These boundaries keep important sequences together, including a base plus combining marks, emoji joined with zero-width joiners, and paired regional indicators. They do not claim to count glyphs, terminal columns, or language-specific words.

```
familyEmoji = "□□□□"
text = "A" || familyEmoji || "B"

say rxunicode..graphemeCount(text)
say rxunicode..graphemeSubstr(text, 2, 1)
say rxunicode..graphemePos(familyEmoji, text)
say rxunicode..graphemeReverse(text)
```

Procedure	Contract
`graphemeCount(text)`	Count default extended grapheme clusters.
`graphemeSubstr(text, start[, length[, pad]])`	Select from a one-based grapheme position. With `length`, right-pad to that many graphemes.
`graphemePos(needle, haystack[, start])`	Find an exact occurrence whose beginning and end are grapheme boundaries; return its one-based grapheme position or zero.
`graphemeReverse(text)`	Reverse cluster order while preserving codepoint order inside each cluster.

`graphemeSubstr` follows the Rexx `SUBSTR` shape. `start` must be positive and a supplied `length` must be non-negative. The default pad is one blank. A supplied pad may contain several codepoints but must form exactly one grapheme cluster. Without `length`, the result extends to the end and is not padded.

Search is exact: neither argument is normalized or folded. A sequence occurring only inside a cluster is not a match, and canonically equivalent but differently encoded text remains different unless normalized separately.

14.6.1 Indexed grapheme snapshot

Direct calls suit one count, bounded slice, search, or reverse. Repeated indexed work should prepare one immutable snapshot:

```
view = .rxunicode..graphemes(text)

say view.text()
say view.count()
say view.at(2)
say view.substr(2, 3)
say view.pos(needle)
say view.reverse()
say view.codepointStart(2)
say view.codepointLength(2)
say view.version()
say view.profile()
```

Method	Contract
`text()`	Return the captured text.
`count()`	Return the grapheme count.
`at(position)`	Return one grapheme at a one-based position.
`substr(start[, length[, pad]])`	Apply the direct substring contract using the retained index.
`pos(needle[, start])`	Apply the direct exact-search contract using the retained index.
`reverse()`	Reverse the retained text by cluster.
`codepointStart(position)`	Return the cluster's one-based `.string` codepoint position.
`codepointLength(position)`	Return the cluster length in `.string` codepoints.
`iterator()`	Return a fresh forward iterator.
`version()`	Return `17.0.0`.
`profile()`	Return `UAX29-C1-1`.

The snapshot owns a packed boundary index proportional to the number of boundaries. Its iterator implements `.StringIterator`:

```
iterator = view.iterator()
do while iterator.hasNext()
  cluster = iterator.next()
  say iterator.index(), iterator.codepointStart(),
      iterator.codepointLength(), cluster
end
call iterator.reset()
```

`index()` returns zero before the first item and then the current one-based grapheme position. Position methods require a current item. Invalid indexed access and `next()` after exhaustion signal `OUT_OF_RANGE`.

14.7 Encoding and decoding

The whole-value codec boundary is typed and strict by default:

```
encode(text = .string[, encoding = "UTF-8"[, replacement = .binary]]) =
     .binary
decode(data = .binary[, encoding = "UTF-8"[, replacement = .string]]) =
     .string
isDecodable(data = .binary[, encoding = "UTF-8"]) = .boolean
isEncodingSupported(encoding = .string) = .boolean
```

The supported canonical encodings are:

Canonical name	Accepted aliases
`UTF-8`	case variations; hyphens and underscores are ignored, for example `utf_8`
`UTF-16LE`, `UTF-16BE`	case variations with optional hyphens or underscores
`UTF-32LE`, `UTF-32BE`	case variations with optional hyphens or underscores
`US-ASCII`	`ASCII`, `ISO-646-US`
`ISO-8859-1`	`latin1`, `ISOLatin1`, `IBM819`, `CP819`, `819`
`Windows-1252`	`Windows1252`, `CP1252`, `1252`
`IBM437`	`CP437`, `437`
`IBM850`	`CP850`, `850`
`IBM1047`	`CP1047`, `1047`

Hyphens and underscores are ignored, surrounding whitespace is trimmed, and matching is case-insensitive. Internal blanks are not accepted. Ambiguous endian names such as `UTF-16` and `UTF-32` are not accepted.

`encode` never adds a byte-order mark. `decode` never consumes one as metadata: a BOM encoded in the selected format becomes U+FEFF in the resulting text. Callers that own a protocol-level BOM convention must add, remove, or interpret it explicitly.

The strict form signals `UNICODE_ERROR` for malformed input or text that the target cannot represent:

```
utf16 = rxunicode..encode(text, "UTF-16LE")
text = rxunicode..decode(utf16, "UTF-16LE")

if rxunicode..isDecodable(bytes, "UTF-8") then
  text = rxunicode..decode(bytes)
```

An explicitly supplied third argument opts into replacement. Decode replacement is non-empty `.string` text and is inserted once per maximal malformed input subpart. Encode replacement is the exact non-empty `.binary` byte sequence to emit for each unmappable scalar, and those bytes must themselves be valid in the target encoding.

```
text = rxunicode..decode("C08041"x as .binary, "UTF-8", "?")  /* ??A */

question = "3F"x as .binary
bytes = rxunicode..encode("Tea □", "Windows-1252", question)

ebcdicQuestion = "6F"x as .binary
bytes = rxunicode..encode(text, "IBM1047", ebcdicQuestion)
```

Replacement bytes are target bytes, not UTF-8 spelling. Thus ASCII and Windows-1252 use byte `3F` for `?`, while IBM1047 uses byte `6F`. Supplying an empty or invalid replacement signals `INVALID_ARGUMENTS`. Omitting the third argument is not the same as supplying an empty replacement.

Windows-1252 follows the retained TUTOR mapping policy for its five undefined byte positions: they decode to the matching C1 control codepoints and can be encoded back. Every byte value round-trips for all four table-backed legacy encodings.

14.7.1 Whole-file conversion

`readbinary(path)` and `writebinary(path, data)` are Level B whole-file byte operations. They make the simplest file conversion explicit:

```
options levelg
import rxfnsb
import rxunicode

source = readbinary("legacy.txt")
text = rxunicode..decode(source, "IBM850")
text = rxunicode..toNFC(text)
target = rxunicode..encode(text, "UTF-8")
written = writebinary("unicode.txt", target)
```

`readbinary` preserves every byte, including embedded NUL and malformed UTF-8. `writebinary` replaces the file with exactly the supplied bytes and returns the number written. Both signal `NOTREADY` for file failures. They load or write a complete value; incremental codecs and encoded stream adapters are not part of this baseline.

14.8 Errors, mutation, and ownership

- Unsupported encoding names and invalid replacement values signal `INVALID_ARGUMENTS`.
- Strict codec conversion failures signal `UNICODE_ERROR`.
- Invalid grapheme arguments signal `INVALID_ARGUMENTS` or `OUT_OF_RANGE` as described above.
- Corrupt generated data or an internal invariant failure signals `FAILURE`.
- Public operations do not mutate their arguments or expose generated mapping tables.
- `.string` is valid UTF-8 text; `.binary` remains arbitrary bytes. Codec conversion does not create a mutable cross-type alias.

14.9 For TUTOR users

TUTOR is an important catalogue of Unicode problems and Rexx-friendly vocabulary. CREXX shares names where the complete semantics align while retaining its statically typed `.string`/`.binary` foundation.

Topic	TUTOR vocabulary	CREXX baseline
Bytes and valid text	BYTES and several text string kinds	`.binary` for bytes; valid UTF-8 `.string` for text
Codepoint operations	CODEPOINTS values/conversions	ordinary `.string` operations count codepoints
Grapheme operations	GRAPHEMES values/conversions	explicit procedures and immutable `.graphemes` snapshot
Normalization	`toNFD`, `toNFC`, `toNFKD`, `toNFKC`, and predicates	the same direct names; never implicit
Default full case fold	`toCasefold`	the same direct name, plus explicitly named simple and Turkic modes
Default case mapping	Unicode upper/lower services	`toUppercase` and `toLowercase`; no titlecase/locale surface in this baseline
Encoding boundary	`ENCODE`, `DECODE`, registry aliases	typed `encode(.string)->.binary` and `decode(.binary)->.string` with selected aliases
Mapping inputs	TUTOR Format A mapping files	Format A accepted at build time; CREXX emits its own checked runtime image
Malformed input	replacement-oriented defaults are available	strict by default; typed replacement is explicit
Encoded streams	encoding can be associated with classic streams	whole-value codecs plus binary file I/O; incremental adapters are roadmap work
Generic dispatch	`UNICODE(text, operation)`	no heterogeneous dispatch BIF in the baseline; direct typed procedures are authoritative
Implicit normalized TEXT	TEXT can carry normalized semantics	no implicit normalization; ordinary equality remains exact
Literal/string-kind suffixes	Y/P/G/T/U forms	not added; existing typed values and explicit services are retained

For a port:

1. Type byte-bearing values as `.binary` and text values as `.string`.

2. Keep the shared normalization, fold, encode, and decode names when their error and endian contracts match.
3. Replace a repeated-index GRAPHEMES conversion with `.rxunicode..graphemes(text);` use direct functions for one-off work.
4. Make any TUTOR implicit-normalization or replacement dependency explicit.
5. Supply exact target bytes for encode replacement, especially for EBCDIC.

This alignment shares source vocabulary without importing dynamic string-kind coercion, mutable global Unicode policy, or hidden transformations into the CREXX language core. A later thin TUTOR-compatibility facade remains possible only for operations whose types, Unicode version, endian behavior, indexing unit, and failure policy are exactly the same; it would delegate to this typed surface rather than become a second semantic authority.

14.10 Deliberate baseline boundary

The module does not yet promise incremental codecs, encoded stream adapters, Unicode properties/names, normalized-caseless key profiles, security profiles, collation, word or sentence boundaries, locale tailoring, or display width. These remain separate designs because each needs its own error, version, indexing, and ownership contract.

The Unicode algorithm appendix describes the prepared data, algorithms, complexity, constant ownership, and performance evidence behind this surface.

15

Packed Numeric Owners

The `rxfnsg` library provides `.packedfloat` and `.packedint` as explicit Level G owners of contiguous host-native numeric values. They wrap the Level B `<packed..float>` and `<packed..int>` binary access surface; they are not new runtime representations and do not change ordinary numeric arrays.

```
options levelg
import rxfnsg

values = .packedfloat(3)
call values.set(0, 1.25)
call values.set(2, 100.0)
say values.get(2)

indexes = .packedint(8)
call indexes.fill(-1)
```

Both classes have the same zero-based API:

Member	Contract
`*(size)`	Allocate `size` items and initialize every byte to zero. Negative or unrepresentable sizes raise `OUT_OF_RANGE`.
`fromBinary(data)`	Own a copy of the host-native bytes. A length that is not a whole number of items raises `INVALID_ARGUMENTS`.
`size()`	Return the number of native items, not the byte length.

Member	Contract
`get(index)`	Read the item at a zero-based index. An incomplete or out-of-range item raises `OUT_OF_RANGE`.
`set(index, value)`	Replace an existing item without resizing. An invalid index raises before mutation.
`resize(size)`	Preserve retained items, discard truncated items, and zero-fill growth.
`fill(value)`	Set every existing item to the supplied native value.
`binary()`	Return a weak mutable `reference .binary` to the owned bytes. The owner must outlive every use of the reference.

`.packedfloat` stores exact VM `.float` (`rxfloat`) values and `.packedint` stores exact VM `.int` (`rxinteger`) values. The object owns those bytes directly in `register.0.binary`. A native provider with a declared `.packedfloat` or `.packedint` argument can therefore bind the object's payload without asking Rexx to call `binary()` or create an intermediate binary value. `fromBinary` is an inbound copy boundary. For ordinary Rexx callers, `payload = dereference values.binary()` creates a scoped live alias without copying, while `payload = snapshot values.binary()` explicitly makes an owned copy. A raw native payload pointer must not be retained across a call or resize.

`rxstats` is the first standard bulk consumer. Its native procedures accept a `.packedfloat` owner directly and borrow `register.0.binary` read-only for the call without an intermediate reference, object copy, or element boxing. `rxstats` does not accept `.packedint`: integer statistics require a separate accumulation-precision decision rather than byte reinterpretation.

`rxvector` borrows `.packedfloat` matrices and queries plus `.packedint` identities through the same zero-copy call boundary. Its explicit `decodef32le` and `encodef32le` procedures are the conversion boundary between portable stored float32 bytes and host-native computation; a packed owner is never itself a portable file or wire encoding.

The representation uses the current host's width, byte order, alignment, and numeric representation. It is suitable for in-process numerical kernels and native providers, but not for files, persistence, network protocols, or cross-host exchange. Use the encoded <at..type> binary-memory surface for those cases.

Release 1 does not provide packed strings or objects, bracket indexing such as `values[2]`, or automatic packed storage for ordinary `.int[]` and `.float[]` arrays. Those require separate post-release language and compatibility work.

PART IV

Environments

16

Address Environments

A command is a simple mechanism for sending a message to some functional unit external to a REXX program. It is usually a request for some service or action, and consists of a single character string. Many operating systems and other programs have a command language interface of this nature.[7]

The *crexxsaa* facility enables the implementation of addressing environments. These can adress functionality using external programs, in the style of Classic Rexx *subcommand handlers*, or implement macro functionality for applications.

Some Address Environments, like `address crexx`, `address shell`, and others, are part of the language core. Other environments can be best classified as libraries; regardless, all are packaged as libraries. The supported `rxsqlite_address` Level G module implements `address sqlite` over CREXX's generic typed `rxsqlite` provider. It delivers an embedded SQL interface without introducing a second native SQLite driver.

Other documented addressing environments are:

- SQLite: a supported core provider with an optional Level G Embedded SQL façade[8];
- CMS: a demonstration of sending commands to an environment which simulates a VM/CMS host;
- THE: The Hessling Editor, a close approximation of the VM/CMS XEDIT

[7] Cowlishaw 1985

[8] Embedded SQL is a style of database access from programs which included blocks of SQL directly into the source code of programs, so without functions calls. This is the standard approach for COBOL, PL/I and some other languages.

editor[9];

[9] including the capability to execute CREXX macro's in THE editor.

PART V

Native Libraries

17 About native libraries

CREXX distribution packages include selected native providers with both dynamic VM and native-static delivery. In particular, the rxsqlite provider includes its pinned SQLite implementation, while `rxsqlite_address` is an optional Level G façade over it. Other integrations can still require a separately installed product library. ODBC and GTK, for example, depend on native facilities supplied for the target platform.

When these libraries are not delivered with the language product itself, there is a distinct possibility of API skew between the CREXX part of the library and the products themselves. This means that a newer version of the external product library has impact on the functioning of the CREXX library API. Insulation between the upgrade paths is always a prudent strategy here, for example by using a virtual machine or a container image. This approach is not necessary when using the CREXX API's proper, for which the contract is guaranteed.

Usage of the native implementations of these parts of the library can offer great benefits to the CREXX application developer, and enables CREXX to participate in a multi-language project by connecting to the same infrastructure. With installation and maintenance of third party libraries there is a larger investment required than with the core library; for this reason, every update of the library documentation will indicate with which release of the external interface libraries the verification has taken place.

Some of the available (procedural or object oriented) libraries are included in the main distributions and other might require a CREXX build from source with special options. The documentation indicates when this is needed. The development team is aware of the fact that this is an even deeper level of investment; on the other hand, there are clear instructions for building the system in the *Programming Guide* and there is a certain satisfaction that goes with building one's own tools.

The different repository systems like *brew*, *apt* and *chocolatey* can deliver assistance in installing external libraries in the required releases.

18

Binary hashing with `rxhash`

The standard `rx_hash` native provider supplies collision-resistant hashing to Level B and Level G code. It is delivered by default but is not part of the minimal compiler or VM core. The provider is process-reentrant and retains no mutable global hashing context.

18.1 SHA-256 procedures

```
rxhash.sha256(data = .binary) = .binary
rxhash.sha256hex(data = .binary) = .string
rxhash.sha256init() = .binary
rxhash.sha256update(state = .binary, data = .binary) = .binary
rxhash.sha256final(state = .binary) = .binary
rxhash.sha256finalhex(state = .binary) = .string
rxhash.sha256file(path = .string) = .binary
rxhash.sha256filehex(path = .string) = .string
```

`sha256`, `sha256final`, and `sha256file` return exactly 32 raw digest bytes. `sha256hex`, `sha256finalhex`, and `sha256filehex` return exactly 64 canonical lowercase hexadecimal characters. Binary inputs are hashed byte for byte; embedded NUL bytes and byte sequences that are not valid UTF-8 participate normally. Inputs are not changed.

The existing `sha256(data)` procedure remains source- and result-compatible. Use the direct hexadecimal forms when text is required rather than applying `bin2x()` and `lower()` at every call site.

18.2 Incremental state

`sha256init()` creates an opaque immutable `.binary` state. Each `sha256update()` validates and decodes its input, hashes the supplied bytes, and returns a newly encoded state; it never mutates the supplied state. An empty update is valid and returns the same canonical value. Finalization validates and decodes a copy, so `sha256final()` and `sha256finalhex()` may be called repeatedly with identical results and do not consume the state.

The state contains no native pointer, C structure image, padding, or host-endian field. It is an ordinary binary value and is safe to copy, retain, persist, or transfer through the supported task/process binary-value mechanisms. State version 1 has this fixed 152-byte, platform-independent encoding:

Offset	Size	Encoding
0	8	ASCII magic `RXSHA256`.
8	1	Format version `1`.
9	1	Flags; zero in version 1.
10	2	Big-endian header size `56`.
12	8	Total input byte count, unsigned big-endian.
20	32	Eight SHA-256 chaining words, each unsigned big-endian.
52	1	Pending-byte count, from 0 through 63.
53	3	Reserved zero bytes.
56	64	Pending block prefix followed by canonical zero padding.
120	32	SHA-256 integrity digest of bytes 0 through 119.

The decoder requires the exact size, magic, version, header size, zero flags and reserved bytes, canonical zero padding, a total count within SHA-256's 64-bit bit-length limit, and a pending count equal to the total byte count modulo 64. Before any complete block, the chaining words must also equal the SHA-256 initial value. The integrity digest detects accidental state corruption; it is not an authenticity or authorization mechanism.

Malformed, truncated, unsupported-version, structurally inconsistent, and integrity-failing states raise `INVALID_ARGUMENTS` with a stable category in the message. An update that would exceed the SHA-256 length limit raises `OVERFLOW_UNDERFLOW`. No invalid state is passed to the hashing engine.

18.3 File hashing

`sha256file()` and `sha256filehex()` synchronously open `path` in binary mode and hash the exact bytes returned by the filesystem. They use the same incremental engine as the memory and state procedures, read fixed 32 KiB chunks, and keep bounded memory regardless of file size. They never read or concatenate the complete file.

An empty path raises `INVALID_ARGUMENTS`. Open, read, and close failures raise `NOTREADY` and identify the failing stage as `file open failed`, `file read failed`, or `file close failed`. The file is closed after EOF and after a read or length failure; when both reading and closing fail, the read failure is the reported primary failure. A file exceeding SHA-256's representable length raises `OVERFLOW_UNDERFLOW`.

These calls deliberately do not select files, connectors, cancellation policy, application size ceilings, or filesystem authorization. The caller owns those policies and should perform any needed preflight checks before starting the synchronous hash.

18.4 Examples

One-shot raw and hexadecimal hashing works identically in Level B and Level G:

```
options levelb

import rxfnsb
import rxhash

data = "610062ff"x as .binary
raw = rxhash..sha256(data)
hex = rxhash..sha256hex(data)
say binlength(raw)                    /* 32 */
say hex                               /* 64 lowercase hexadecimal
    characters */
```

Incremental callers retain the returned state after every update:

```
state = rxhash..sha256init()
state = rxhash..sha256update(state, "6162"x as .binary)
state = rxhash..sha256update(state, ''x as .binary)
state = rxhash..sha256update(state, "63"x as .binary)

say rxhash..sha256finalhex(state)
/* ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad */
```

File hashing is a single synchronous call:

```
digest = rxhash..sha256file("artifacts/index.bin")
say rxhash..sha256filehex("artifacts/index.bin")
```

18.5 Delivery

The functions are declared directly by RXPA, so there is no REXX wrapper to import. A retained call records provider `rx_hash` in the compiled image. Normal `rxvm`, `rxbvm`, and `rxtvm` execution resolves the installed dynamic provider automatically; `crexx -native` selects its static archive automatically. An explicit provider filename or `-p` argument is not required for a standard installation.

The same provider also publishes binary-safe named 32-bit functions:

Procedure	Contract
`djb2(data = .binary)`	DJB2 modulo 2^32.
`murmur3(data = .binary, seed = .int)`	MurmurHash3 x86 32-bit with the supplied seed.
`fnv1a(data = .binary)`	FNV-1a 32-bit.
`crc32(data = .binary)`	IEEE CRC-32.

They return the unsigned 32-bit bit pattern as a non-negative `.int`. Embedded zero bytes participate normally. These are checksums or table/container hashes, not substitutes for SHA-256 where collision resistance is required. The historical `rxmath` aliases are intentionally absent in this pre-release API.

19

Statistics with `rxstats`

`rxstats` is the process-reentrant Level G native provider for bulk statistics over `.packedfloat` owners. The provider borrows their host-native payloads read-only for each call: it does not box elements, call `binary()`, copy the buffer, or retain a payload pointer.

Procedure	Result
`mean(values = .packedfloat)`	Arithmetic mean; at least one value.
`stddev(values = .packedfloat)`	Sample standard deviation; at least two values.
`covariance(x = .packedfloat, y = .packedfloat)`	Sample covariance; equal lengths of at least two.
`correlation(x = .packedfloat, y = .packedfloat)`	Pearson correlation; equal lengths of at least two and non-zero variance.
`regression(x = .packedfloat, y = .packedfloat)`	Immutable `.linearfit`; equal lengths of at least two and non-zero x variance.

```
options levelg floats_binary
import rxfnsg
import rxstats

values = .packedfloat(3)
call values.set(0, 1.0)
call values.set(1, 2.0)
call values.set(2, 3.0)
say rxstats..mean(values)
```

`.linearfit` is a small immutable REXX value class in the `rxstats` namespace. Its public factory accepts slope and intercept values, and its `slope()` and `intercept()`

methods expose the named results. Its packed payload remains private; the class is only a result carrier, not a second Rexx implementation of the statistics algorithms.

The implementation uses Neumaier-compensated shifted-origin sums for means, second moments, covariance, and regression. An ill-conditioned central moment triggers a compensated second pass around the computed mean. This preserves the accepted historical results on the retained `1e12` cancellation probe. Every input value must be finite. NaN, infinity, invalid counts, unequal paired lengths, and undefined zero-variance operations raise `INVALID_ARGUMENTS` with a procedure-specific message. A finite input whose calculation cannot be represented as a finite native float raises `OVERFLOW_UNDERFLOW`. Passing a bare, uninitialized `.packedfloat` raises `OBJECT_NOT_INITIALIZED`.

The surface deliberately does not accept `.packedint`, raw `.binary`, boxed `.float[]`, ranges, offsets, strides, weights, or missing-value masks. Integer accumulation needs a separately approved precision contract. Views, vector operations, and accelerator selection belong to the later `rxvector` design, which must reuse the same packed storage boundary.

RXBIN provider metadata selects `rxstats.rxplugin` automatically for ordinary execution and `rxstats.a` or the platform-equivalent archive for `crexx -native`. No Rexx declaration wrapper or manual plugin list is needed.

20 Identifier generation with `rxid`

The optional Level G `rxid` native provider supplies six zero-argument string generators:

Procedure	Result
`uuid4()`	36-character random UUID with version 4 and RFC variant bits.
`uuid7()`	36-character time-ordered UUID with version 7 and RFC variant bits.
`ulid()`	26-character time-ordered Crockford Base32 ULID.
`nanoid()`	21-character URL-safe Nano ID.
`snowflake()`	Decimal text representation of a time/node/sequence identifier.
`base58()`	Bitcoin-alphabet Base58 text generated from random bytes.

```
options levelg
import rxid

request_id = rxid..uuid4()
ordered_id = rxid..uuid7()
```

UUIDv4, UUIDv7, ULID, Nano ID, and Base58 randomness comes from the platform cryptographically secure random source. UUIDv7 and ULID generation serialize their monotonic same-millisecond state; Snowflake maintains its own synchronized sequence state. A platform random-source failure raises `FAILURE`.

The pre-release private `id._*`, `rxuuid`, and `idlib` draft names have been retired. Import `rxid` and use the public names above. The provider is installed in dynamic and static forms and is found automatically from RXBIN dependency metadata; no REXX wrapper or explicit VM plugin list is required.

21 Filesystem operations with `rxfs`

The standard/default `rxfs` provider owns filename and directory conveniences. It is callable from Level B source when installed, but it is not part of the minimal bootstrap core.

Procedure	Contract
`cwd()`	Current working directory.
`loadpath()`	Directory containing the running executable.
`chdir(path)`	Change directory.
`isdir(path)` / `isfile(path)`	Return 1 when the path has the requested type, otherwise 0.
`mkdir(path)` / `rmdir(path)`	Create or remove one directory.
`delete(path)` / `rename(source, target)`	Delete or rename a file.
`listdir(path, expose entries)`	Populate a `.string[]` and return its entry count.
`append(source, target)`	Append source bytes to target and return the byte count.

Mutating functions retain the historical integer status convention: zero is success where no count is returned; negative values distinguish missing, permission, general I/O, open, and write failures. Invalid arity or an unrepresentably long path raises `INVALID_ARGUMENTS`.

`rxfs` replaces the filesystem subset of the retired broad `system` plugin. The `crexx` compiler driver now links only `rxfs`, its actual native provider dependency. Dynamic execution and native packaging resolve it automatically from RXBIN provider metadata.

22

Host information and timing with `rxplatform`

The optional Level G `rxplatform` provider contains the narrow portable host-information and timing subset extracted from the retired `system` plugin:

Procedure	Contract
`uptime()`	Host uptime in seconds.
`user()`	Current operating-system user name.
`host()`	Host name.
`osname()`	Portable family name such as `Windows`, `Linux`, or `macOS`.
`sleep(milliseconds)`	Sleep for 0 through 4,294,967,295 milliseconds and return zero.
`message(text)`	Display an informational desktop dialog and return zero.
`confirm(text)`	Display a Yes/No desktop dialog; return `1` for Yes and `0` for No.
`input(prompt)`	Request one text line; return entered text or an empty string on Cancel.
`directory(prompt [, initial])`	Select a directory, optionally starting at `initial`; return its path or an empty string on Cancel.
`file(prompt [, save [, initial]])`	Open or save a file; return its path or an empty string on Cancel.

`message()` and `confirm()` use native Windows dialogs on Windows, `osascript` on macOS, and `zenity` or `kdialog` on Linux. Linux therefore requires one of those desktop dialog helpers to be installed. They are synchronous and return only after the user closes the dialog. If the platform cannot provide a desktop dialog, they raise `FAILURE`.

The class library `Os` wrapper also provides:

Method	Contract
`x.warning(text)`	Display the text as an informational dialog prefixed with `Warning:`.
`x.error(text)`	Display the text as an informational dialog prefixed with `Error:`.

Both wrapper methods return zero on success and use the same platform dialog support as `x.message()`.

Unavailable host information raises `FAILURE`; invalid sleep values raise `INVALID_-ARGUMENTS`. Clipboard, beep, process-global values, pipe operations, and developer inspection are intentionally not included. The process-reentrant provider ships in automatic dynamic and static forms under provider ID `rxplatform`.

23 Graphical User Interfaces

Release status: this page describes the optional GTK plugin area. It is useful as implementation reference material, but it is not part of the Release 1 beta baseline language contract. Verify the plugin build option and tests before depending on these APIs in a packaged release.

23.1 The GTK Plugin

The (multiplatform) GTK Plugin enables a straightforward way to implement portable REXX programs with a graphical user interface.

The GTK plugin now includes an experimental same-thread callback event-loop proof of concept. It is not background multitasking. The plugin remains a build-time option.[10]

23.2 Overview

This section provides an overview of the optional, experimental GTK 3 plugin for cREXX. `lib/plugins/gui/gui.c` implements the RXPA provider, `gui_sample1.crexx` is the larger demonstration, and `gui_test.crexx` is the focused automated/interactive contract test.

[10] `-DENABLE_GTK=ON`

LevelG Classes

23.3 Quick Start Guide

23.3.1 Basic Window Creation

Creating a basic window is the first step in building a GUI application. Here's a minimal example:

```
init_window(title,width,height)
it_window("My First GUI", 400, 300) \= 1 then exit 1
call add_text "GTK is active", 20, 20
ok_button = add_button("OK", 90, 80)
cancel_button = add_button("Cancel", 210, 80)

call show_window -1, -1

do forever
    event = process_events(250)
    if event < 0 | event = ok_button | event = cancel_button then leave
end
call cleanup_gui
```

`process_events(timeout)` returns a positive widget handle for an event, `0` when the interval expires without an event, and `-1` when the main window is closed. A timeout is an idle poll, not an instruction to exit.

The callback alternative is `run_event_loop(handler)`. The handler implements `.eventhandler.on_event(widget=.int,event_type=.string) -> .int`, returning non-zero to continue and zero to leave the loop. It first receives the deterministic event `(0, "ready")`; GTK then reports `clicked`, `close`, `changed`, and `selected` events. Callbacks are synchronous on the GTK/VM thread, may call GUI functions, and remain borrowed only until the event-loop call returns. The polling API remains supported.

23.3.2 Adding Common Widgets

Here's how to add basic widgets to your window:

```
/* Add a label */
text1 = add_text("Enter your name:", 10, 10)

/* Add an input field */
input1 = add_edit(10, 40, 200)

/* Add a button */
```

```
button1 = add_button("Submit", 10, 70)

/* Add a list */
list1 = add_list(10, 100, 200, 100)  /* x, y, width, height */
call list_add_item list1, "Item 1", "lightblue"
call list_add_item list1, "Item 2", "lightgreen"

/* Add a combo box */
items.1 = "Option 1"
items.2 = "Option 2"
items.3 = "Option 3"
combo1 = add_combo(items, 10, 220)
```

23.3.3 Complete Working Example

Here's a complete example that demonstrates common GUI patterns:

```
/* Initialize GUI */
call init_window "Contact Form", 300, 400

/* Add form fields */
call add_text "Name:", 10, 10
name_field = add_edit(10, 30, 260)

call add_text "Email:", 10, 70
email_field = add_edit(10, 90, 260)

call add_text "Type:", 10, 130
type.1 = "Personal"
type.2 = "Business"
type.3 = "Other"
type_combo = add_combo(type, 10, 150)

/* Add buttons */
submit = add_button("Submit", 10, 200)
clear = add_button("Clear", 120, 200)

/* Add status area */
status = add_text("Ready", 10, 350)

/* Show window */
call show_window -1, -1

/* Main event loop */
do forever
    event = process_events(500)

    if event < 0 then leave              /* Window closed */
```

```
    else if event = submit then do       /* Submit button clicked */
        name = get_edit(name_field)
        email = get_edit(email_field)
        if name = "" | email = "" then
            call set_text status, "Please fill all fields"
        else
            call set_text status, "Form submitted!"
    end
    else if event = clear then do        /* Clear button clicked */
        call set_edit name_field, ""
        call set_edit email_field, ""
        call set_text status, "Form cleared"
    end
end
call cleanup_gui
```

23.3.4 Common Patterns and Best Practices

GUI Layout Guidelines

1. **Widget Spacing**
   - Leave 10-20 pixels between widgets
   - Group related widgets together
   - Align widgets for visual consistency
2. **Layout Organization**
   - Place important controls at the top
   - Group related controls together
   - Use consistent spacing and alignment
   - Consider tab order for input fields
3. **Size Guidelines**
   - Make clickable elements at least 25x25 pixels
   - Leave enough space for text in labels
   - Consider window resizing behavior

Error Handling Strategies

1.
```
/* Validate required fields */
if input_text = "" then do
    call set_status status_bar, "Please fill required field"
    call set_sensitive submit_button, 0
```

```
    return
end

button = add_button("Click Me", 10, 10)
if button < 0 then do
    call notify_pick "Error", "Failed to create button", "error"
    return
end
```

3.
```
rc = copy_file(source, target)
if rc \= 0 then do
    call notify_pick "Error", "Failed to copy file", "error"
    return
end
```

Common Usage Patterns

1. **Enable/Disable Controls**

```
/* Disable submit until valid */
call set_sensitive submit_button, 0

/* Enable when valid */
if valid_input then
    call set_sensitive submit_button, 1

/* Show operation status */
call set_status status_bar, "Processing..."
/* ... do work ... */
call set_status status_bar, "Complete"
```

3.
```
/* Show splash during long operation */
call splash_pick "Working", "Please wait...", 0, 300, 100
/* ... do work ... */
call cleanup_gui
```

Performance Tips

1. **Event Processing**
   - Use appropriate timeout values in process_events
   - Avoid lengthy operations in event loop
   - Consider using background processing for long tasks
2. **Memory Management**
   - Clean up resources when no longer needed

    - Use cleanup_gui when closing application
    - Free any allocated memory
3. **Widget Updates**
    - Batch multiple updates together
    - Avoid unnecessary widget refreshes
    - Use efficient data structures

23.4 Installation Instructions

To set up the environment for running the GUI application, follow these steps:

23.4.1 For Linux

1. **Install GTK**: Ensure your system has GTK installed. You can install it using your package manager. For example:

    - ```
      # Debian/Ubuntu
      sudo apt-get install unixodbc unixodbc-dev

      # RedHat/CentOS
      sudo yum install unixODBC unixODBC-devel

      # For MySQL
      sudo apt-get install libmyodbc   # Debian/Ubuntu
      sudo yum install mysql-connector-odbc   # RedHat/CentOS
      ```

      ```
      sudo dnf install gtk3-devel
      ```

    - ```
      brew install gtk+3
      ```

2. **Configure and build cREXX**:

    ```
    cmake -S . -B cmake-build-debug -G Ninja -DCMAKE_BUILD_TYPE=
        Debug -DENABLE_GTK=ON
    cmake --build cmake-build-debug --target gui_test
    ctest --test-dir cmake-build-debug -R '^gui_test_(noopt|opt)$'
        --output-on-failure
    ```

23.4.2 For Windows

1. **Install GTK**: Download the GTK installer for Windows from the GTK website. Follow the instructions to install GTK on your system.
2. **Set Environment Variables**: After installation, you may need to set the `PATH` environment variable to include the GTK `bin` directory. This allows you to run GTK applications from the command line.
3. **Build cREXX**: Configure the CREXX CMake project with `-DENABLE_GTK=ON`. CMake owns the RXPA module name, include paths, and GTK link dependencies; `gui.c` is not a standalone executable.

23.5 CREXX GUI Documentation

23.5.1 Function Quick Reference

23.5.2 Window and Widget Functions

INIT_WINDOW

```
init_window(title,width,height)
it_window("My First GUI", 400, 300) \= 1 then exit 1
call add_text "GTK is active", 20, 20
ok_button = add_button("OK", 90, 80)
cancel_button = add_button("Cancel", 210, 80)

call show_window -1, -1

do forever
    event = process_events(250)
    if event < 0 | event = ok_button | event = cancel_button then leave
end
call cleanup_gui
```

- **Description**: Initializes the main application window and sets up the GTK environment.
- **Parameters**:
  - `title`: Title of the window.
  - `width`: Width of the window.

    - `height`: Height of the window.
- **Returns**: An integer indicating success (1) or failure (0).

ADD_BUTTON

`add_button(button_text,x-offset,y-offset)`

- **Description**: Creates a new button with the specified text and adds it to the fixed container. The button is displayed at the specified coordinates. At present, the button size is fixed at 100x25 pixels, but this might be changed in the future.
- **Parameters**:
    - `button_text`: The text to display on the button (e.g., `"Click Me"`).
    - `x`: The X-coordinate for button placement (e.g., `10`).
    - `y`: The Y-coordinate for button placement (e.g., `10`).
- **Returns**:
    - The index of the button in the global widgets array.
    - Returns `-1` if the button cannot be created.
- **Example**:

```
button_index = add_button("Submit", 50, 100)
if button_index < 0 then
    say "Error: Could not add button."
```

ADD_TEXT

`add_text(text,x-offset,y-offset)`

- **Description**: Creates a new label and adds it to the fixed container. At present, the label size is fixed at 100x25 pixels, but this might be changed in the future.
- **Parameters**:
    - `text`: Text to display on the label.
    - `x`: X-coordinate for label placement.
    - `y`: Y-coordinate for label placement.
- **Returns**: The index of the label in the global widgets array.
- **Example**:

```
label_index = add_text("Hello World", 10, 50)
```

ADD_COMBO

`add_combo(item-list,x-offset,y-offset)`

- **Description**: Creates a new combo box and adds it to the fixed container. At present, the combo box size is fixed at 100x25 pixels, but this might be changed in the future.
- **Parameters**:
  - `item-list`: An array containing a list of items to add to the combo box (e.g., `["Option 1", "Option 2", "Option 3"]`).
  - `x`: X-coordinate for combo box placement.
  - `y`: Y-coordinate for combo box placement.
- **Returns**: The index of the combo box in the global widgets array.
- **Example**:

```
combo_index = add_combo(["Option 1", "Option 2", "Option 3"], 10,
    100)
```

ADD_LIST

`add_list(x-offset,y-offset,width,height)`

- **Description**: Creates a new list box and adds it to the fixed container. At present, the list box size is fixed at 100x25 pixels, but this might be changed in the future.
- **Parameters**:
  - `x`: X-coordinate for list placement.
  - `y`: Y-coordinate for list placement.
  - `width`: Width of the list box.
  - `height`: Height of the list box.
- **Returns**: The index of the list in the global widgets array.
- **Example**:

```
list_index = add_list(10, 150, 200, 100)
```

ADD_EDIT

`add_edit(x-offset,y-offset,width)`

- **Description**: Creates a new editable text field (entry) and adds it to the fixed container.
- **Parameters**:
  - `x`: X-coordinate for entry placement.
  - `y`: Y-coordinate for entry placement.
  - `width`: Requested width in pixels.
- **Returns**: The index of the entry in the global widgets array.
- **Example**:

```
edit_index = add_edit(10, 100, 240)
call set_edit edit_index, "Type here..."
current_text = get_edit(edit_index)
```

LIST_ADD_ITEM

`list_add_item(list,x-offset,y-offset,bg_colour)`

- **Description**: Adds a new item to the specified list.
- **Parameters**:
  - `list`: Index of the list to which the item will be added.
  - `text`: Text to display for the new item.
  - `bg_colour`: Background colour for the item (refer to the X11 colours for available colours or use RGB in hexadecimal notation, e.g., `#RRGGBB`).
- **Returns**: An integer indicating success (1).
- **Example**:

```
call list_add_item(list_index, "Item 1", "lightblue")
```

LIST_GET_SELECTED

`list_get_selected(list)`

- **Description**: Retrieves the index of the currently selected item in the specified list.

- **Parameters**:
  - `list`: Index of the list to check for selection.
- **Returns**: The index of the selected item or -1 if no selection is made.
- **Example**:

```
selected_index = list_get_selected(list_index)
```

LIST_GET_SELECTED_ITEM

```
list_get_selected_item(list)
```

- **Description**: Retrieves the text of the currently selected item in the specified list.
- **Parameters**:
  - `list`: Index of the list to check for selection.
- **Returns**: The text of the selected item or an empty string if no selection is made.
- **Example**:

```
selected_text = list_get_selected_item(list_index)
```

LIST_SET_HEADER

```
call list_set_header list_index, "My List Header", "black", "
    lightgray"
```

- **Description**: Sets a header for the specified list.
- **Parameters**:
  - `list`: Index of the list to set the header for.
  - `header_text`: Text to display as the header.
  - `text_colour`: Text colour for the header.
  - `bg_colour`: Background colour of the header (refer to the X11 colours for available colours or use RGB in hexadecimal notation).
- **Returns**: An integer indicating success (1).
- **Example**:

```
call list_set_header list_index, "My List Header", "black", "
    lightgray"
```

CLEANUP_GUI

```
cleanup_gui
```

- **Description**: Cleans up the main window, widget and graph registries, and held GTK display/settings references. `cleanup` remains a compatibility alias.
- **Returns**: None.
- `call cleanup_gui`

GET_EDIT

```
get_edit(index)
```

- **Description**: Returns the current text from an edit field created by `add_edit`.
- **Returns**: The edit text, or an empty string for an invalid or non-edit handle.
- **Example**:

```
value = get_edit(edit_index)
```

GET_WIDGET_ADDRESS

```
get_widget_address(index)
```

- **Description**: Retrieves the address of a widget based on its index in the global widgets array.
- **Parameters**:
  - `index`: Index of the widget to retrieve.
- **Returns**: The address of the widget or 0 if the index is invalid.
- **Example**:

```
widget_address = get_widget_address(button_index)
```

RGB_TO_HEX

```
rgb_to_hex(r, g, b)
```

- **Description**: Converts RGB values to a hexadecimal color string.
- **Parameters**:

    - `r`: Red component (0-255).
    - `g`: Green component (0-255).
    - `b`: Blue component (0-255).
- **Returns**: A string representing the color in hexadecimal format (e.g., `#RRGGBB`). Returns an empty string if the input values are out of range.
- **Example**:

```
hex_color = rgb_to_hex(255, 0, 0)  // Returns "#FF0000"
```

SET_SENSITIVE

```
set_sensitive(widget_index, sensitive)
```

- **Description**: Enables or disables a widget while keeping it visible.
- **Parameters**:
    - `widget_index`: Index of the widget to modify.
    - `sensitive`: 1 to enable the widget, 0 to disable it.
- **Returns**:
    - `0` on success
    - `-1` if the widget index is invalid
- **Example**:

```
/* Disable a widget */
call set_sensitive button_index, 0

/* Enable a widget */
call set_sensitive button_index, 1
```

COPY_FILE

```
copy_file(source_path, destination_path)
```

- **Description**: Copies a file from the source path to the destination path.
- **Parameters**:
    - `source_path`: Path to the source file.
    - `destination_path`: Path where the file should be copied to.
- **Returns**:
    - `0` on success

    - `-8` on failure
- **Example**:

```
rc = copy_file("C:\source\file.txt", "C:\destination\file.txt")
if rc = 0 then
    say "File copied successfully"
else
    say "Error copying file"
```

23.5.3 Dialog and Picker Services

FILE_PICK

```
file_pick(title, initial_dir, save_dialog, pattern)
```

- **Description**: Opens a file picker dialog to select or save a file.
- **Parameters**:
    - `title`: Title of the dialog window.
    - `initial_dir`: Initial directory to open.
    - `save_dialog`: 0 for open dialog, 1 for save dialog.
    - `pattern`: File pattern to filter (e.g., ".*txt*", ".rexx").
- **Returns**: Selected file path as string, or empty string if cancelled.
- **Examples**:

```
/* Basic file selection */
filepath = file_pick("Select a File", "C:\Documents", 0, "*.rexx")

/* Save dialog with specific extension */
savepath = file_pick("Save As", "C:\Documents", 1, "*.txt")

/* Multiple file patterns */
files = file_pick("Select Document", "C:\Docs", 0, "*.doc;*.docx;*.
    txt")

/* All files */
anyfile = file_pick("Select Any File", "C:\Files", 0, "*.*")

filepath = file_pick("Select File", initial_dir, 0, "*.rexx")
if filepath = "" then do
    say "User cancelled file selection or an error occurred"
    return
end
```

initial directory for better user experience - Use clear, descriptive titles - Specify file patterns to help users find relevant files - Handle empty returns (user cancellation)

PATH_PICK

`path_pick(title, initial_dir)`

- **Description**: Opens a directory picker dialog to select a folder.
- **Parameters**:
  - `title`: Title of the dialog window.
  - `initial_dir`: Initial directory to open.
- **Returns**: Selected directory path as string, or empty string if cancelled.
- **Examples**:

```
/* Basic directory selection */
dirpath = path_pick("Select Folder", "C:\Documents")

/* Project directory selection with validation */
projectDir = path_pick("Select Project Directory", "C:\Projects")
if projectDir \= "" then do
    if directory_exists(projectDir) then
        say "Selected directory:" projectDir
    else
        say "Invalid directory selection"
end

dirpath = path_pick("Select Directory", initial_dir)
if dirpath = "" then do
    say "User cancelled directory selection or an error occurred"
    return
end
```

output directories - Validate directory existence after selection - Consider combining with FILE_PICK for complete file operations

DATE_PICK

`date_pick(title, show_time, format)`

- **Description**: Opens a calendar dialog to select a date and optionally time.
- **Parameters**:

- `title`: Title of the dialog window.
- `show_time`: 1 to include time selection, 0 for date only.
- `format`: Date/time format string.

- **Returns**: Selected date/time as formatted string, or empty string if cancelled.
- **Format Strings**:
  - `YYYY`: Four-digit year
  - `MM`: Two-digit month (01-12)
  - `DD`: Two-digit day (01-31)
  - `HH`: Hours (00-23)
  - `mm`: Minutes (00-59)
  - `ss`: Seconds (00-59)
- **Examples**:

```
/* Date only selection */
date = date_pick("Select Date", 0, "YYYY-MM-DD")

/* Date and time selection */
datetime = date_pick("Select Date and Time", 1, "YYYY-MM-DD HH:mm:ss"
    )

/* Custom format */
custom = date_pick("Select Date", 0, "DD/MM/YYYY")

/* With validation */
selected_date = date_pick("Select Delivery Date", 1, "YYYY-MM-DD HH:
    mm")
if selected_date \= "" then do
    if validate_future_date(selected_date) then
        say "Valid future date selected:" selected_date
    else
        say "Please select a future date"
end

date = date_pick("Select Date", 0, "YYYY-MM-DD")
if date = "" then do
    say "User cancelled date selection or an error occurred"
    return
end
```

strings for your locale - Consider time zones when using time selection - Validate dates for specific requirements (future dates, working days, etc.) - Use consistent date formats throughout your application

LIST_PICK

`list_pick(title, items, multi_select, message)`

- **Description**: Opens a dialog with a list of items to select from.
- **Parameters**:
  - `title`: Title of the dialog window.
  - `items`: Array of items to display in the list.
  - `multi_select`: 1 to allow multiple selections, 0 for single selection.
  - `message`: Optional message to display above the list.
- **Returns**: Selected item(s) as string (comma-separated if multiple), or empty string if cancelled.
- **Examples**:

```
/* Single selection */
options = ["Small", "Medium", "Large"]
size = list_pick("Select Size", options, 0, "Choose your preferred
    size:")

/* Multiple selection */
toppings = ["Cheese", "Pepperoni", "Mushrooms", "Olives", "Onions"]
selected = list_pick("Select Toppings", toppings, 1, "Choose your
    toppings:")

/* With validation */
if selected \= "" then do
    parse var selected first ',' rest
    say "Primary selection:" first
    if rest \= "" then
        say "Additional selections:" rest
end

/* Check for empty selection */
result = list_pick("Select Items", items, 1, "Please select:")
if result = "" then do
    say "No selection made"
    return
end

/* Handle multiple selections */
if pos(',', result) > 0 then do
    say "Multiple items selected"
    do while result \= ""
        parse var result item ',' result
        say "Selected:" item
    end
```

```
end
```

reasonably sized (consider scrolling for long lists) - Provide clear instructions in the message parameter - Consider sorting items alphabetically - Use meaningful item names - Handle both single and multiple selection cases

COMBO_PICK

`combo_pick(title, message, items)`

- **Description**: Shows a dialog with a combo box for selection.
- **Parameters**:
  - `title`: Title of the dialog window.
  - `message`: Message to display above the combo box.
  - `items`: Array of items to display in the combo box.
- **Returns**: Selected item as string, or empty string if cancelled.
- **Example**:

```
items = ["Option 1", "Option 2", "Option 3"]
selected = combo_pick("Choose Option", "Please select:", items)
```

TREE_PICK

`tree_pick(title, message, items, parents, multi_select)`

- **Description**: Shows a tree view dialog for hierarchical selection.
- **Parameters**:
  - `title`: Title of the dialog window.
  - `message`: Message to display above the tree.
  - `items`: Array of items to display in the tree.
  - `parents`: Array indicating parent-child relationships.
  - `multi_select`: 1 to allow multiple selections, 0 for single selection.
- **Returns**: Selected item(s) as string (comma-separated if multiple), or empty string if cancelled.
- **Examples**:

```
/* Basic tree structure */
items.1 = "Root"
items.2 = "Child1"
```

```
items.3 = "Child2"
items.4 = "Subchild1"
parents.1 = ""
parents.2 = "Root"
parents.3 = "Root"
parents.4 = "Child1"
selected = tree_pick("Select Items", "Choose from tree:", items,
    parents, 0)

/* File system example */
items.1 = "Documents"
items.2 = "Work"
items.3 = "Personal"
items.4 = "Reports"
items.5 = "Letters"
parents.1 = ""
parents.2 = "Documents"
parents.3 = "Documents"
parents.4 = "Work"
parents.5 = "Personal"
files = tree_pick("File Browser", "Select files:", items, parents, 1)

/* Organization chart */
items.1 = "CEO"
items.2 = "HR"
items.3 = "IT"
items.4 = "HR Manager"
items.5 = "HR Staff"
items.6 = "IT Manager"
items.7 = "Developers"
parents.1 = ""
parents.2 = "CEO"
parents.3 = "CEO"
parents.4 = "HR"
parents.5 = "HR"
parents.6 = "IT"
parents.7 = "IT"
positions = tree_pick("Organization", "Select department:", items,
    parents, 0)

if selected = "" then do
    say "No selection made or dialog cancelled"
    return
end

/* Handle multiple selections */
do while selected \= ""
    parse var selected item ',' selected
    say "Processing:" item
end
```

reasonable (3-4 levels maximum for usability) - Use clear parent-child relationships - Consider using icons for different types of nodes - Provide clear hierarchy in item names - Handle both single and multiple selection modes appropriately

INPUT_PICK

```
input_pick(title, message, default_value, password_mode)
```

- **Description**: Shows an input dialog for text entry.
- **Parameters**:
  - `title`: Title of the dialog window.
  - `message`: Message to display above the input field.
  - `default_value`: Default text in the input field.
  - `password_mode`: 1 to mask input (for passwords), 0 for normal text.
- **Returns**: Entered text as string, or empty string if cancelled.
- **Example**:

```
labels = ["Name:", "Email:", "Phone:"]
defaults = ["John Doe", "john@example.com", ""]
result = form_pick("User Information", "Please enter your details:",
    labels, defaults)
```

FORM_PICK

```
labels = ["Name:", "Email:", "Phone:"]
defaults = ["John Doe", "john@example.com", ""]
result = form_pick("User Information", "Please enter your details:",
    labels, defaults)
```

- **Description**: Shows a form dialog with multiple input fields.
- **Parameters**:
  - `title`: Title of the dialog window.
  - `message`: Message to display above the form.
  - `labels`: Array of labels for each input field.
  - `defaults`: Array of default values for each input field.
- **Returns**: Comma-separated string of entered values, or empty string if cancelled.

- ```
  labels = ["Name:", "Email:", "Phone:"]
  defaults = ["John Doe", "john@example.com", ""]
  result = form_pick("User Information", "Please enter your
      details:", labels, defaults)
  ```

PAGE_PICK

```
page_pick(title, pages, labels, defaults)
```

- **Description**: Shows a multi-page form dialog.
- **Parameters**:
  - `title`: Title of the dialog window.
  - `pages`: Array of page titles.
  - `labels`: Array of labels for input fields on each page.
  - `defaults`: Array of default values for input fields.
- **Returns**: Comma-separated string of entered values, or empty string if cancelled.
- **Example**:

```
dialog_pick(title, message, buttons)
eferences"]
  labels = ["Name,Age", "Email,Phone", "Theme,Language"]
  defaults = ["John Doe,30", "john@example.com,555-0123", "Dark,English
     "]
  result = page_pick("User Profile", pages, labels, defaults)
```

DIALOG_PICK

```
dialog_pick(title, message, buttons)
eferences"]
  labels = ["Name,Age", "Email,Phone", "Theme,Language"]
  defaults = ["John Doe,30", "john@example.com,555-0123", "Dark,English
     "]
  result = page_pick("User Profile", pages, labels, defaults)
```

- **Description**: Shows a message dialog with custom buttons.
- **Parameters**:
  - `title`: Title of the dialog window.
  - `message`: Message to display.
  - `buttons`: Array of button labels.
```

- **Returns**: Index of clicked button (1-based), or 0 if cancelled.
- ```
  buttons = ["Yes", "No", "Cancel"]
  response = dialog_pick("Confirm", "Do you want to save?",
      buttons)
  ```

NOTIFY_PICK

```
notify_pick(title, message, type)
```

- **Description**: Shows a notification/message dialog.
- **Parameters**:
  - `title`: Title of the notification.
  - `message`: Message to display.
  - `type`: Type of notification ("info", "warning", "error", "question").
- **Returns**: Empty string.
- **Example**:

```
call notify_pick "Success", "Operation completed successfully", "info
    "
call notify_pick "Warning", "Low disk space", "warning"
```

SPLASH_PICK

```
splash_pick(title, message, duration, width, height, image_path)
```

- **Description**: Shows a splash screen with optional image.
- **Parameters**:
  - `title`: Title of the splash window.
  - `message`: Message to display.
  - `duration`: Display duration in seconds (0 for manual close).
  - `width`: Width of the splash window.
  - `height`: Height of the splash window.
  - `image_path`: Path to an image file to display (optional).
- **Returns**: Empty string.
- **Example**:

```
/* Show splash screen for 3 seconds */
call splash_pick "Welcome", "Loading...", 3, 400, 300, "logo.png"
```
```

TEXT_DISPLAY

`text_display(title, message, item_text)`

- **Description**: Shows a simple text display dialog.
- **Parameters**:
  - `title`: Title of the dialog window.
  - `message`: Message to display above the text.
  - `item_text`: Text content to display (can include newlines).
- **Returns**: Empty string.
- **Example**:

```
text = "First line\nSecond line\nThird line"
call text_display "Document", "Content:", text
```

TEXT_DISPLAY_PICK

`text_display_pick(title, message, item_texts)`

- **Description**: Shows a dialog with formatted text display.
- **Parameters**:
  - `title`: Title of the dialog window.
  - `message`: Message to display above the text.
  - `item_texts`: Array of text items to display.
- **Returns**: Empty string.
- **Example**:

```
texts = ["First paragraph", "Second paragraph", "Third paragraph"]
call text_display_pick "Document", "Preview:", texts
```

COLOR_PICK

`color_pick(title, message, default_color)`

- **Description**: Shows a color picker dialog.
- **Parameters**:
  - `title`: Title of the dialog window.
  - `message`: Message to display above the color picker.

    - `default_color`: Initial color (in hex format or X11 color name).
- **Returns**: Selected color in hex format (#RRGGBB), or empty string if cancelled.
- **Example**:

```
color = color_pick("Choose Color", "Select background color:", "#
    FF0000")
```

TREE_DIAGRAM

```
tree_diagram(items, parents)
```

- **Description**: Shows a graphical tree diagram.
- **Parameters**:
    - `items`: Array of items to display in the diagram.
    - `parents`: Array indicating parent-child relationships.
- **Returns**: Empty string.
- **Example**:

```
items = ["CEO", "Manager1", "Manager2", "Employee1", "Employee2"]
parents = ["", "CEO", "CEO", "Manager1", "Manager2"]
call tree_diagram items, parents
```

ADD_GRAPH

```
add_graph(x, y, width, height, function_type,x1,y1,x2,y2,x3,y3)
```

- **Description**: Creates a widget to display mathematical functions and graphs.
- **Parameters**:
    - `x`: X-coordinate for graph placement.
    - `y`: Y-coordinate for graph placement.
    - `width`: Width of the graph area.
    - `height`: Height of the graph area.
    - `x1`: x-value-1.
    - `y1`: y-value related to x-value-1.
    - `x2`: x-value-2.
    - `y2`: y-value related to x-value-2.
    - `x3`: x-value-3.

  - `y3`: y-value related to x-value-3. A maximum of three individual graph elements can be added to a single graph widget. By default, these graphs are hidden and must be triggered using the upd_graph function to be displayed. The passed x/y values must be float arrays. The sizing of the x/y axes is determined by the x1/y1 arrays. Therefore, it is essential that the ranges of x2/y2 and x3/y3 align properly.
- **Returns**: The index of the graph widget in the global widgets array.
- **Example**:

```
graph1 = add_graph(10, 10, 300, 200,x1,y1,x2,y2,x3,y3)
```

size for visibility (recommended minimum 200x200) - Consider window size when placing graphs - Multiple graphs can be added to the same window

ADD_R2CHART

```
add_r2chart(x, y, width, height, function_type,x1,y1,x2,y2,x3,y3)
```

- **Description**: Creates a widget to display ⊠^{2}⊠ Chart – Specifies only the first quadrant (non-negative real numbers). Can display up to three graphs in the same widget.
- **Parameters**:
  - `x`: X-coordinate for graph placement.
  - `y`: Y-coordinate for graph placement.
  - `width`: Width of the graph area.
  - `height`: Height of the graph area.
  - `x1`: Array of x coordinates for the first graph.
  - `y1`: Array of y coordinates for the first graph.
  - `x2`: Array of x coordinates for the second graph (optional).
  - `y2`: Array of y coordinates for the second graph (optional).
  - `x3`: Array of x coordinates for the third graph (optional).
  - `y3`: Array of y coordinates for the third graph (optional).
- **Returns**: The index of the graph widget in the global widgets array.
- **Example**:

```
/* Create arrays for the graphs */
j=1
do i=-20 to 20
```

```
    x.j=i/10
    y.j=x.j*x.j    /* Quadratic function */
    j=j+1
  end

  /* Create graph widget with two functions */
  graph = add_r2chart(10, 30, 570, 550, x, y, a, b)

  /* Update graph colors */
  call update_graph graph, 1, 1, 3, "yellow"  /* First graph */
  call update_graph graph, 2, 1, 3, "blue"    /* Second graph */
```

23.5.4 UPDATE_GRAPH

`update_graph(graph_index, graph_num, line_type, line_width, color)`

- **Description**: Updates the appearance of a specific graph within a graph widget. Since there may be more than one graph in the widget, this function only prepares the appearance of the specified graph. This is because the drawing function operates on the entire widget, which contains all the graphs. To complete the drawing process, you can use a value of -1 for the graphnum parameter.
- **Parameters**:
  - `graph_index`: Index of the graph widget.
  - `graph_num`: Which graph to update (1-3) or -1 to perform the draw process.
  - `line_type`: Type of line (1=solid, 2=dots).
  - `line_width`: Width of the line/dots in pixels.
  - `color`: Color name (X11 color) or hex value.
- **Returns**: None.
- **Example**:

```
/* Initialize GUI */
call init_window "Application Settings", 500, 600

/* Create tabs or sections */
call add_text "General Settings", 10, 10

/* Theme Selection */
call add_text "Theme:", 10, 40
themes.1 = "Light"
themes.2 = "Dark"
```

```
themes.3 = "System Default"
theme_combo = add_combo(themes, 10, 60)

/* Language Selection */
call add_text "Language:", 10, 100
langs.1 = "English"
langs.2 = "Spanish"
langs.3 = "French"
lang_combo = add_combo(langs, 10, 120)

/* File Paths */
call add_text "Default Save Location:", 10, 160
path_field = add_edit(10, 180, 380)
path_btn = add_button("Browse", 400, 180)

/* Preferences */
call add_text "Preferences", 10, 220
auto_save = add_checkbox("Auto-save files", 10, 250)
show_toolbar = add_checkbox("Show toolbar", 10, 280)
enable_logging = add_checkbox("Enable logging", 10, 310)

/* Advanced Settings */
call add_text "Advanced", 10, 350
buffer_size = add_edit(10, 380, 100)
call add_text "Buffer Size (KB)", 120, 385

/* Action Buttons */
save_btn = add_button("Save Settings", 10, 500)
cancel_btn = add_button("Cancel", 120, 500)
default_btn = add_button("Restore Defaults", 230, 500)

/* Status Bar */
status_bar = add_status_bar()

/* Show window */
call show_window -1, -1

/* Main event loop */
do forever
    event = process_events(500)
    if event < 0 then leave

    else if event = path_btn then do
        new_path = path_pick("Select Default Save Location", "C:\")
        if new_path \= "" then
            call set_edit path_field, new_path
    end

    else if event = save_btn then do
```

```
        /* Validate settings */
        buffer = get_edit(buffer_size)
        if \datatype(buffer, 'W') | buffer < 1 then do
            call notify_pick "Error", "Invalid buffer size", "error"
            iterate
        end

        /* Save settings */
        if save_config() then do
            call set_status status_bar, "Settings saved successfully"
            call notify_pick "Success", "Settings have been saved", "
                info"
            leave
        end
    end

    else if event = default_btn then do
        if dialog_pick("Confirm", "Restore default settings?", "Yes|No"
           ) = 1 then
            call restore_defaults
    end

    else if event = cancel_btn then
        leave
end

/* Helper function to save configuration */
save_config: procedure
    /* Implementation of save logic */
return 1

/* Helper function to restore defaults */
restore_defaults: procedure
    call set_edit buffer_size, "1024"
    call set_sensitive auto_save, 1
    call set_sensitive show_toolbar, 1
    call set_sensitive enable_logging, 0
return
```

23.5.5 Common Issues with Graphs

1. **Graph Not Updating**: Ensure that the correct graph index is being used in `update_graph`. Each graph should maintain its own state.
2. **Graphs Overlapping**: Make sure to set proper coordinates and sizes for each graph to avoid overlap.

23.5.6 Debugging Tips

1. **Debug Output**: Use debug logging to trace function calls and variable states.
2. **Event Processing**: Ensure the event loop is running to allow for updates and redraws.
3. **Widget States**: Verify that widget states are being managed correctly, especially when updating or redrawing.

23.5.7 Configuration Dialog

Here's an example of a settings/configuration dialog:

```
/* Initialize GUI */
call init_window "Application Settings", 500, 600

/* Create tabs or sections */
call add_text "General Settings", 10, 10

/* Theme Selection */
call add_text "Theme:", 10, 40
themes.1 = "Light"
themes.2 = "Dark"
themes.3 = "System Default"
theme_combo = add_combo(themes, 10, 60)

/* Language Selection */
call add_text "Language:", 10, 100
langs.1 = "English"
langs.2 = "Spanish"
langs.3 = "French"
lang_combo = add_combo(langs, 10, 120)

/* File Paths */
call add_text "Default Save Location:", 10, 160
path_field = add_edit(10, 180, 380)
path_btn = add_button("Browse", 400, 180)

/* Preferences */
call add_text "Preferences", 10, 220
auto_save = add_checkbox("Auto-save files", 10, 250)
show_toolbar = add_checkbox("Show toolbar", 10, 280)
enable_logging = add_checkbox("Enable logging", 10, 310)

/* Advanced Settings */
call add_text "Advanced", 10, 350
buffer_size = add_edit(10, 380, 100)
```

```
call add_text "Buffer Size (KB)", 120, 385

/* Action Buttons */
save_btn = add_button("Save Settings", 10, 500)
cancel_btn = add_button("Cancel", 120, 500)
default_btn = add_button("Restore Defaults", 230, 500)

/* Status Bar */
status_bar = add_status_bar()

/* Show window */
call show_window -1, -1

/* Main event loop */
do forever
    event = process_events(500)
    if event < 0 then leave

    else if event = path_btn then do
        new_path = path_pick("Select Default Save Location", "C:\")
        if new_path \= "" then
            call set_edit path_field, new_path
    end

    else if event = save_btn then do
        /* Validate settings */
        buffer = get_edit(buffer_size)
        if \datatype(buffer, 'W') | buffer < 1 then do
            call notify_pick "Error", "Invalid buffer size", "error"
            iterate
        end

        /* Save settings */
        if save_config() then do
            call set_status status_bar, "Settings saved successfully"
            call notify_pick "Success", "Settings have been saved", "
               info"
            leave
        end
    end

    else if event = default_btn then do
        if dialog_pick("Confirm", "Restore default settings?", "Yes|No"
           ) = 1 then
            call restore_defaults
    end

    else if event = cancel_btn then
        leave
```

```
end

/* Helper function to save configuration */
save_config: procedure
    /* Implementation of save logic */
return 1

/* Helper function to restore defaults */
restore_defaults: procedure
    call set_edit buffer_size, "1024"
    call set_sensitive auto_save, 1
    call set_sensitive show_toolbar, 1
    call set_sensitive enable_logging, 0
return
```

23.6 Troubleshooting

23.6.1 Common Issues

1. **Window Not Showing**
   - Check if `show_window` was called
   - Verify window dimensions are reasonable
   - Ensure GTK is properly initialized
2. **Widget Creation Failures**

```
button = add_button("Click Me", 10, 10)
if button < 0 then do
    call notify_pick "Error", "Failed to create button", "error"
    return
end
```

3. **Event Processing Issues**
   - Ensure event loop is properly implemented
   - Check event token values
   - Verify widget indices
4. **Memory Issues**
   - Call `cleanup_gui` when closing
   - Don't exceed maximum widget count (128)
   - Free resources properly

23.6.2 Debugging Tips

1. **Event Debugging**

```
/* Add debug output */
do forever
    event = process_events(500)
    say "Event Token:" event
    if event < 0 then leave
end
```

2. **Widget State Verification**

```
/* Check widget status */
widget_addr = get_widget_address(widget_index)
if widget_addr = 0 then
    say "Widget not found or invalid"
```

3. **Status Updates**

```
/* Add status messages */
call set_status status_bar, "Processing event:" event
```

23.6.3 Known Limitations

1. **Widget Limits**
   - Maximum 128 widgets per window
   - Fixed widget sizes (100x25 pixels for most widgets)
   - Limited layout options (fixed positioning only)
2. **Event Handling**
   - Single event loop per window
   - No asynchronous event processing
   - Limited event types
3. **Platform Specific**
   - Some features may work differently on different platforms
   - Font and color rendering may vary
   - File paths need platform-specific handling

23.6.4 X11 Colours

This section provides a list of commonly used X11 colours that you can use in your GTK application. You can specify these colours by name or use their RGB values in hexadecimal notation (e.g., `#RRGGBB`).

23.6.5 Basic Colours

- **`black`**: Represents the color black.
- **`white`**: Represents the color white.
- **`red`**: Represents the color red.
- **`green`**: Represents the color green.
- **`blue`**: Represents the color blue.
- **`yellow`**: Represents the color yellow.
- **`cyan`**: Represents the color cyan.
- **`magenta`**: Represents the color magenta.

23.6.6 Shades of Gray

- **`gray`**: A medium shade of gray.
- **`lightgray`**: A lighter shade of gray.
- **`darkgray`**: A darker shade of gray.
- **`dimgray`**: A dimmer shade of gray.
- **`slategray`**: A bluish-gray color.
- **`light slate gray`** : A light bluish-gray color.

24 The CREXX Curses library

Release status: this page is a placeholder for optional terminal UI support. It is not part of the Release 1 beta baseline language contract.

25

ODBC Programming

25.1 Overview

The ODBC plugin provides a bridge between CREXX and database systems through ODBC (Open Database Connectivity). It supports both SQL databases and CSV files.

Each current VM receives an independent ODBC session containing its environment, connection, default statement, prepared statements, transaction state and last diagnostic. Calls from different VM threads can therefore use different connections concurrently. Statement IDs are opaque, belong to the creating session and become invalid at close, disconnect, reconnect or VM teardown. Older hosts that do not understand RXPA sessions use the plugin's process-default compatibility session and should continue to serialize their calls.

25.2 Quick Start

```
/* Basic database query */
rc = odbc_connect("myDB", "user", "pass")
if rc < 0 then exit

/* Explicitly set the database */
rc = odbc_database("CompanyDB")
if rc <> 0 then exit

rc = odbc_execute("SELECT * FROM mytable")
```

```
if rc < 0 then do
    say "Error:" odbc_error_message()
    exit
end

do while odbc_fetch() = 0
    value = odbc_getcolumn(1)
    say value
end

call odbc_disconnect
```

25.3 Functions

25.3.1 Connection Management

odbc_connect

```
rc = odbc_connect(dsn, username, password)
```

database or CSV file. - **Parameters:** - dsn: Database name or directory path for CSV - username: Database username (empty for CSV) - password: Database password (empty for CSV) - **Returns:** - 0: Success - -1: Environment handle allocation failed - -2: Environment configuration failed - -3: Connection handle allocation failed - -4: Connection failed

odbc_disconnect

```
call odbc_disconnect
```

connection and frees resources.

25.3.2 Query Execution

odbc_execute

```
rc = odbc_execute(sql)
```

Parameters: - sql: SQL query string - **Returns:** - 0: Success - -1: Statement handle allocation failed - -2: Query execution failed

Prepared statements

`odbc_prepare(sql)` returns a positive opaque statement ID. Negative results are errors: `-1` means no connection, `-2` a host allocation failure, `-3` a driver prepare failure and `-4` statement-ID exhaustion.

Bind parameters by one-based position before execution:

```
statement = odbc_prepare("select name from employee where id = ? and
    active = ?")
if statement <= 0 then exit

if odbc_bind_integer(statement, 1, 42) <> 0 then exit
if odbc_bind_string(statement, 2, "Y") <> 0 then exit
if odbc_execute_prepared(statement) <> 0 then exit

do while odbc_fetch_statement(statement) = 0
    say odbc_getcolumn_statement(statement, 1)
end

if odbc_close_prepared(statement) <> 0 then exit
```

The bind functions are `odbc_bind_string`, `odbc_bind_integer`, `odbc_bind_float` and `odbc_bind_null`. They return `0` on success and `-1` for an invalid statement/position or driver bind failure. A failed rebind leaves the previous successful binding valid. The current RXPA string accessor does not expose a byte length, so binary parameter binding is intentionally not offered; use a driver-supported textual encoding where appropriate.

`odbc_execute_prepared(statement)` returns `0`, `-1` for an invalid ID, or `-2` for a driver execution failure. `odbc_reset_prepared(statement)` closes the current result and removes all bindings while preserving the prepared SQL. `odbc_close_prepared(statement)` releases the statement permanently.

Multiple prepared statements may be active in one session. Use these statement-aware result functions instead of the default-statement spellings:

- `odbc_fetch_statement(statement)`
- `odbc_columns_statement(statement)`
- `odbc_getcolumn_statement(statement, column)`
- `odbc_colname_statement(statement, column)`
- `odbc_coltype_statement(statement, column)`
- `odbc_column_info_statement(statement, column)`
- `odbc_row_count_statement(statement)`
- `odbc_move_to_statement(statement, row)`
- `odbc_error_message_statement(statement)`

25.3.3 Result Set Navigation

odbc_fetch

```
rc = odbc_fetch()
```

result set. This function should be called in a loop to process each row of the result set.

- `0`: a row is available
- `1`: no more rows
- `-1`: driver error

odbc_getcolumn

```
value = odbc_getcolumn(column)
```

specific column in the current row. - **Parameters:** - column: Column number (1-based) - **Returns:** - Column value as string - "-1" on error

25.3.4 Metadata Functions

odbc_columns

```
cols = odbc_columns()
```

the result set. - **Returns:** - Number of columns - -1 on error

odbc_colname

```
name = odbc_colname(column)
```

column. - **Parameters:** - column: Column number (1-based) - **Returns:** - Column name - Empty string on error

odbc_coltype

```
type = odbc_coltype(column)
```

column. - **Parameters:** - column: Column number (1-based) - **Returns:** - SQL type code (see SQL Data Types section) - -1 on error

25.3.5 Transaction Management

odbc_begin_transaction

```
call odbc_begin_transaction
```

transaction by turning off autocommit. - **Returns:** - 0: Success - -1: Failed to set autocommit off

odbc_commit

```
call odbc_commit
```

- **Returns:** - 0: Success - -1: Commit failed

odbc_rollback

```
call odbc_rollback
```

- **Returns:** - 0: Success - -1: Rollback failed

25.3.6 Error Handling

odbc_error_message

```
message = odbc_error_message()
```

ODBC. - **Returns:** Detailed error message string

25.3.7 Diagnostic Functions

odbc_get_diagnostics

```
info = odbc_get_diagnostics()
```

information about the last error. - **Returns:** String containing: - SQLSTATE code - Native error code - Detailed message

odbc_column_info

```
info = odbc_column_info(column)
```

about a column. - **Parameters:** - column: Column number (1-based) - **Returns:** Comma-delimited `name,type,size,decimals,nullable`, or an empty string on error.

25.3.8 Example Usage

```
/* Get column information */
do col = 1 to odbc_columns()
    info = odbc_column_info(col)
```

```
    parse var info name','type','size','dec','null
    say "Column" col":" name
    say "  Type:" type
    say "  Size:" size
    say "  Decimals:" dec
    say "  Nullable:" null
end

/* Error handling with diagnostics */
if rc < 0 then do
    say "Error occurred:"
    say odbc_get_diagnostics()
    exit
end
```

25.4 CSV File Support

25.4.1 Connecting to CSV Files

```
/* Connect to a directory containing CSV files */
rc = odbc_connect("C:\path\to\csv\directory", "", "")
if rc < 0 then exit

/* Query CSV file */
rc = odbc_execute("SELECT * FROM [mydata.csv]")
```

25.4.2 CSV-Specific Notes

- Leave username and password empty for CSV connections
- The dsn parameter should be the directory containing CSV files
- CSV files should use semicolon (;) as delimiter
- First row can contain column headers
- File names are used as table names in queries
- Use square brackets around file names in queries: [filename.csv]

25.5 SQL Data Types

Common SQL types returned by odbc_coltype:

TABLE 2: ODBC SQL Data Types

Constant	Value	Description
SQL_CHAR	1	Fixed-length character string
SQL_NUMERIC	2	Exact numeric value with precision and scale
SQL_DECIMAL	3	Exact numeric value with precision and scale
SQL_INTEGER	4	32-bit integer
SQL_SMALLINT	5	16-bit integer
SQL_FLOAT	6	Approximate numeric value
SQL_REAL	7	Single precision floating point
SQL_DOUBLE	8	Double precision floating point
SQL_TYPE_TIMESTAMP	11	Date and time
SQL_VARCHAR	12	Variable-length character string
SQL_WVARCHAR	-9	Unicode variable-length character string
SQL_WCHAR	-8	Unicode fixed-length character string
SQL_WLONGVARCHAR	-10	Unicode long variable-length character string

25.6 Usage Examples

25.6.1 Basic Query Example

```
/* Connect to database */
rc = odbc_connect("myDB", "user", "pass")
if rc < 0 then do
    say "Connection failed:" odbc_error_message()
    exit
end

/* Explicitly set the database */
rc = odbc_database("CompanyDB")
if rc <> 0 then do
    say "Failed to set database:" odbc_error_message()
    exit
end

/* Execute query */
rc = odbc_execute("SELECT * FROM employees")
if rc < 0 then do
    say "Query failed:" odbc_error_message()
    call odbc_disconnect
    exit
end

/* Get and display column information */
cols = odbc_columns()
```

```
do col = 1 to cols
    name = odbc_colname(col)
    type = odbc_coltype(col)
    say "Column" col":" name "Type:" type
end

/* Fetch and display results */
do while odbc_fetch() = 0
    do col = 1 to cols
        value = odbc_getcolumn(col)
        say "Column" col":" value
    end
    say "------------------"
end

call odbc_disconnect
```

25.6.2 Transaction Example

```
/* Start transaction */
call odbc_begin_transaction

/* First operation */
rc = odbc_execute("UPDATE employees SET salary = salary * 1.1")
if rc < 0 then do
    say "Update failed:" odbc_error_message()
    call odbc_rollback
    call odbc_disconnect
    exit
end

/* Second operation */
rc = odbc_execute("INSERT INTO audit_log VALUES ('Salary update')")
if rc < 0 then do
    say "Audit failed:" odbc_error_message()
    call odbc_rollback
    call odbc_disconnect
    exit
end

/* Commit if both operations succeeded */
call odbc_commit
call odbc_disconnect
```

25.6.3 Advanced Usage Examples

1. **Table and Schema Information**

   ```
   /* Get list of tables */
   ```

```
tables = odbc_tables()
do while tables \= ''
    parse var tables table ';' tables
    say "Found table:" table
end

/* Get primary keys for a table */
keys = odbc_primary_keys("employees")
do while keys \= ''
    parse var keys key ';' keys
    say "Primary key column:" key
end

/* Execute multiple SQL statements */
sql = "UPDATE employees SET salary = salary * 1.1 WHERE department = 'IT';"
sql = sql || "INSERT INTO audit_log VALUES ('Salary update');"
sql = sql || "UPDATE budget SET amount = amount - 50000 WHERE dept = 'IT';"

rc = odbc_execute_batch(sql, ";")
if rc < 0 then do
    say "Batch execution failed:" odbc_error_message()
    exit
end
```

3.
```
/* Get database system info */
dbms = odbc_get_info()
say "Database system:" dbms

/* Get connection attributes */
autocommit = odbc_get_connection_attr(SQL_ATTR_AUTOCOMMIT)
say "Autocommit mode:" autocommit
```

4. **Result Set Navigation**

```
/* Execute a query */
rc = odbc_execute("SELECT * FROM employees ORDER BY salary DESC")

/* Get total rows */
rows = odbc_row_count()
say "Total rows:" rows

/* Move to specific row */
rc = odbc_move_to(5)  /* Move to 5th row */
if rc >= 0 then do
    name = odbc_getcolumn(1)
    salary = odbc_getcolumn(2)
    say "5th highest salary:" name salary
end
```

```
/* Move to last row */
rc = odbc_move_to(rows)
if rc >= 0 then do
    name = odbc_getcolumn(1)
    salary = odbc_getcolumn(2)
    say "Lowest salary:" name salary
end

/* Connect and start transaction */
rc = odbc_connect("MyDB", "user", "pass")
call odbc_begin_transaction

/* Get database info */
say "Connected to" odbc_get_info()

/* Get tables and their primary keys */
tables = odbc_tables()
do while tables \= ''
    parse var tables table ';' tables
    say "Table:" table

    keys = odbc_primary_keys(table)
    do while keys \= ''
        parse var keys key ';' keys
        say "  Primary key:" key
    end
end

/* Execute query with row navigation */
rc = odbc_execute("SELECT * FROM employees")
rows = odbc_row_count()
say "Found" rows "employees"

/* Move through result set */
do row = 1 to rows
    rc = odbc_move_to(row)
    if rc >= 0 then do
        name = odbc_getcolumn(1)
        say "Employee" row":" name
    end
end

call odbc_commit
call odbc_disconnect
```

These examples demonstrate: - Table and schema metadata retrieval - Batch SQL execution - Database system information - Result set navigation - Row counting - Combined usage patterns

Note: Some functions may not be available with all ODBC drivers or database systems. Always check return codes for errors.

26 ODBC Plugin Installation

The OBDC plugin is dependent on a working ODBC installation on the operating platform. This chapter assists in installing ODBC on the main platforms supported by cREXX.

26.1 Windows ODBC Configuration

On Windows, elevated privileges might be required to achieve a working ODBC setup. When planning an application using ODBC in an environment you do not control, like a corporate environment, it is a good idea to involve administrative staff in an early stage to insure that ODBC is installed and working. For simple applications, consider CREXX's bundled SQLite typed provider and ADDRESS environment, which needs no separately installed ODBC driver.

26.1.1 Setting Up ODBC Data Sources

1. **Open ODBC Data Source Administrator**
   - Press Windows + R
   - Type: `odbcad32.exe` for 32-bit or
   - Navigate to: Control Panel > Administrative Tools > ODBC Data Sources
2. **Choose the Correct Administrator**
   - Use 32-bit: `C:\Windows\SysWOW64\odbcad32.exe`
   - Use 64-bit: `C:\Windows\System32\odbcad32.exe`

- Note: Match the bitness of your REXX environment

26.1.2 Configuring Database Connections

1. **MySQL Configuration**
   Driver: MySQL ODBC Driver
   Server: localhost (or your server)
   Database: your_database_name
   User: your_username
   Password: your_password
   Port: 3306 (default)
2. **CSV File Configuration**
   - Driver: `Microsoft Access Text Driver (*.txt, *.csv)`
   - Directory: Path to your CSV files
   - Extensions: CSV
   - Format: Delimited(;)

26.1.3 Required ODBC Drivers

1. **For MySQL**
   - Download MySQL Connector/ODBC from MySQL website
   - Run installer and select correct bitness (32/64)
   - Verify in ODBC Administrator
2. **For CSV Files**
   - Microsoft Access Text Driver (built into Windows)
   - Microsoft Excel Driver (optional, for backup)

26.1.4 Testing the Connection

1. **Using ODBC Administrator**
   - Click "Test Connection" after setup
   - Verify connectivity before using in REXX

```
/* Test connection */
rc = odbc_connect("myDSN", "user", "pass")
if rc < 0 then do
    say "Connection failed:" odbc_error_message()
```

```
    exit
end
say "Connection successful"
call odbc_disconnect
```

26.1.5 Troubleshooting Windows ODBC

1. **Common Issues**
   - Driver not found: Install correct ODBC driver
   - Wrong bitness: Match 32/64-bit versions
   - Path issues: Use double backslashes in paths
   - Permission denied: Check Windows permissions

   ```
   /* List available drivers */
   rc = odbc_connect("", "", "")  /* Will fail but show drivers */
   ```

2.
   ```
   /* CSV directory path formats */
   rc = odbc_connect("C:\\Data\\CSV", "", "")     /* Double
       backslash */
   rc = odbc_connect("C:/Data/CSV", "", "")       /* Forward slash
        */
   ```

26.1.6 Windows-Specific Notes

1. **File Permissions**
   - Ensure read/write access to database/files
   - Run as administrator if needed
   - Check Windows User Account Control (UAC)
2. **System DSN vs. User DSN**
   - System DSN: Available to all users
   - User DSN: Only for current user
   - File DSN: Portable between systems
3. **Registry Settings**
   - ODBC drivers listed in:
     - `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI`
   - DSN configurations in:
     - `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI`
4. **Environment Variables**

- ODBC_PATH: Path to ODBC installation
- ODBCINI: Path to odbc.ini file
- ODBCINST: Path to odbcinst.ini file

26.2 Linux ODBC Configuration

26.2.1 Installing ODBC Components

1. **Install Required Packages**

```
# Debian/Ubuntu
sudo apt-get install unixodbc unixodbc-dev

# RedHat/CentOS
sudo yum install unixODBC unixODBC-devel

# For MySQL
sudo apt-get install libmyodbc   # Debian/Ubuntu
sudo yum install mysql-connector-odbc   # RedHat/CentOS
```

2. **Configuration Files**
   - `/etc/odbc.ini`: System DSN definitions
   - `/etc/odbcinst.ini`: Driver definitions
   - `~/.odbc.ini`: User-specific DSN definitions

26.2.2 Setting Up Drivers

1.
```
# /etc/odbcinst.ini
[MySQL]
Description = MySQL ODBC Driver
Driver = /usr/lib/x86_64-linux-gnu/odbc/libmyodbc.so
Setup = /usr/lib/x86_64-linux-gnu/odbc/libodbcmyS.so
FileUsage = 1

# /etc/odbcinst.ini
[Text]
Description = Text Driver
Driver = /usr/lib/x86_64-linux-gnu/odbc/libtxtS.so
Setup = /usr/lib/x86_64-linux-gnu/odbc/libtxtS.so
FileUsage = 1
```

26.2.3 Configuring Data Sources

1. ```
   # /etc/odbc.ini or ~/.odbc.ini
   [MyDB]
   Driver = MySQL
   Server = localhost
   Database = test
   Port = 3306

   # /etc/odbc.ini or ~/.odbc.ini
   [MyCSV]
   Driver = Text
   Directory = /path/to/csv/files
   Extension = csv
   Delimiter = ;
   ```

26.2.4 Testing Configuration

1. ```
   isql -v MyDB username password
   ```

2. **Using odbcinst** `bash <!--bashtest2.sh--> odbcinst -q -d     # List drivers`
   `odbcinst -q -s    # List data sources`

26.3 macOS ODBC Configuration

26.3.1 Installing ODBC Components

1. ```
   brew install unixodbc
   brew install mysql-connector-odbc  # For MySQL
   ```

2. **Configuration Locations**
   - `/usr/local/etc/odbc.ini`
   - `/usr/local/etc/odbcinst.ini`
   - `~/Library/ODBC/odbc.ini`

26.3.2 Driver Setup

1. ```
   # /usr/local/etc/odbcinst.ini
   [MySQL]
   Description = MySQL ODBC Driver
   ```

```
Driver = /usr/local/lib/libmyodbc5w.so
Setup = /usr/local/lib/libodbcmyS.so

# /usr/local/etc/odbcinst.ini
[Text]
Description = Text Driver
Driver = /usr/local/lib/libtextodbc.so
Setup = /usr/local/lib/libtextodbc.so
```

26.3.3 DSN Configuration

1.
```
# /usr/local/etc/odbc.ini
[MyDB]
Driver = MySQL
Server = localhost
Database = test
Port = 3306

# /usr/local/etc/odbc.ini
[MyCSV]
Driver = Text
Directory = /Users/username/csv
Extension = csv
Delimiter = ;
```

26.3.4 Testing and Troubleshooting

1.
```
export ODBCINI=/usr/local/etc/odbc.ini
export ODBCSYSINI=/usr/local/etc
export ODBCINSTINI=/usr/local/etc/odbcinst.ini

# Test connection
iodbctest "DSN=MyDB;UID=username;PWD=password"

# List drivers
iodbcadm-gtk    # GUI tool
odbcinst -q -d  # Command line
```

26.3.5 Common Issues on Unix-like Systems

1. **Driver Not Found**
   - Check library paths

- Verify driver installation
- Check file permissions

```
# Fix permissions
sudo chmod 644 /etc/odbc.ini
sudo chmod 644 /etc/odbcinst.ini
```

3.
```
# Add to .bashrc or equivalent
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

4. **SELinux Considerations (Linux)**

```
# If using SELinux
sudo setsebool -P allow_odbc_connection 1
```

26.3.6 Platform-Specific Notes

1. **Linux**
   - Different distributions may have different package names
   - Check distribution-specific documentation
   - Consider using package manager's search function
2. **macOS**
   - Homebrew is recommended for package management
   - M1/ARM64 may require different configurations
   - Check architecture-specific paths

26.4 Database-Specific Configurations

26.4.1 SQLite Configuration

1. **Installing SQLite ODBC Driver**

```
# Windows
Download from http://www.ch-werner.de/sqliteodbc/

# Linux (Debian/Ubuntu)
sudo apt-get install libsqliteodbc

# macOS
brew install sqliteodbc
```

```
# Windows: System DSN
[SQLite3 ODBC Driver]
Description=SQLite3 ODBC Driver
Driver=sqliteodbc.dll
Setup=sqliteodbc.dll

# Linux/macOS: /etc/odbcinst.ini
[SQLite3]
Description=SQLite3 ODBC Driver
Driver=/usr/lib/x86_64-linux-gnu/odbc/libsqlite3odbc.so
Setup=/usr/lib/x86_64-linux-gnu/odbc/libsqlite3odbc.so
```

3.

```
[MySQLiteDB]
Driver=SQLite3
Database=/path/to/database.db
Timeout=2000
NoTXN=0
SyncPragma=NORMAL
```

26.4.2 PostgreSQL Configuration

1. **Installing PostgreSQL ODBC Driver**

```
# Windows
Download from postgresql.org/ftp/odbc/versions/

# Linux
sudo apt-get install odbc-postgresql      # Debian/Ubuntu
sudo yum install postgresql-odbc          # RedHat/CentOS

# macOS
brew install psqlodbc

[PostgreSQL]
Description=PostgreSQL ODBC Driver
Driver=psqlodbcw.dll                      # Windows
Driver=/usr/lib/psqlodbcw.so             # Linux
Driver=/usr/local/lib/psqlodbcw.so       # macOS
```

3.

```
[MyPostgres]
Driver=PostgreSQL
Server=localhost
Port=5432
Database=mydb
Username=myuser
Password=mypass
```

26.4.3 Microsoft SQL Server Configuration

1. **Installing MSSQL ODBC Driver**

```
# Windows
Download from Microsoft's website

# Linux
curl https://packages.microsoft.com/keys/microsoft.asc | sudo
    apt-key add -
sudo apt-get install msodbcsql17

# macOS
brew tap microsoft/mssql-release
brew install msodbcsql17
```

```
[MSSQL]
Description=Microsoft SQL Server Driver
Driver={ODBC Driver 17 for SQL Server}
```

3.

```
[MyMSSQL]
Driver=MSSQL
Server=serveraddress,1433
Database=mydatabase
```

26.4.4 MariaDB Configuration

1. **Installing MariaDB ODBC Driver**

```
# Windows
Download from mariadb.org/download/

# Linux
sudo apt-get install libmariadb3 libmariadb-dev
sudo apt-get install mariadb-connector-odbc

# macOS
brew install mariadb-connector-odbc
```

```
[MariaDB]
Description=MariaDB ODBC Driver
Driver=libmaodbc.so
```

26.4.5 Usage Notes for Different Databases

2.
```
/* Connect to SQLite database */
rc = odbc_connect("MySQLiteDB", "", "")  /* No username/
    password needed */

/* Create table example */
rc = odbc_execute("CREATE TABLE IF NOT EXISTS test (id INTEGER
    PRIMARY KEY, name TEXT)")

/* Handle schema */
rc = odbc_execute("SET search_path TO myschema")

/* Use prepared statements */
rc = odbc_execute("PREPARE myplan (int) AS SELECT * FROM
    mytable WHERE id = $1")
```

3.
```
/* Windows Authentication */
rc = odbc_connect("MyMSSQL", "", "")  /* Empty credentials for
    Windows Auth */

/* Handle schemas */
rc = odbc_execute("USE mydatabase; SELECT * FROM dbo.mytable")
```

26.4.6 Common Issues by Database Type

1. **SQLite**
    - File permissions
    - Database file locking
    - Journal file location
2. **PostgreSQL**
    - Schema handling
    - SSL certificate configuration
    - Network connectivity
3. **MSSQL**
    - Windows vs SQL authentication
    - Network protocols
    - Instance naming
4. **MariaDB/MySQL**
    - Character set configuration
    - SSL connection setup
    - Time zone handling

26.5 Security Considerations

1. **Connection Security**
    - Avoid hardcoding credentials in scripts
    - Use environment variables or secure configuration files
    - Enable SSL/TLS where available
2. **File Permissions**
    - Secure database files with appropriate permissions
    - Protect configuration files containing credentials
    - Consider using system DSN instead of file DSN
3. **Network Security**
    - Use encrypted connections when possible
    - Configure firewalls appropriately
    - Consider using VPN for remote connections

26.6 Performance Tips

1. **Query Optimization**
    - Use appropriate WHERE clauses
    - Consider indexing frequently queried columns
    - Avoid SELECT * when possible
2. **Connection Management**
    - Reuse connections when possible
    - Close connections properly
    - Use transactions for bulk operations
3. **Memory Considerations**
    - Handle large result sets in chunks
    - Clear statement handles when done
    - Monitor memory usage with large datasets

26.6.1 Database Selection Functions

odbc_database

```
rc = odbc_database("mydb")        /* Switch current catalogue */
```

catalogue through the ODBC driver.

Note: This function is currently only supported for MySQL databases. For other database systems, use database-specific SQL commands via `odbc_execute()`.

Important: After connecting to a database, you should explicitly set the database using `odbc_database()` before performing any operations.

Parameters: - newdb: Name of the catalogue/database to select

Returns: - 0 if the driver accepted the change - -1 if there is no connection or the driver rejected the change

26.6.2 Primary Key Retrieval

odbc_primary_keys

```
keys = odbc_primary_keys(table)
```

for the specified table.

Returns: - A string containing the primary key column names, separated by spaces. - An error message if the primary keys cannot be retrieved.

PART VI

Appendices

A

cREXX Standard Libraries and Built-In Functions (BIFs)

The `crexx` toolchain implements its Standard Libraries and Built-In Functions (BIFs) using a hybrid approach. Some core functions are implemented natively in C via the **cREXX Plugin Architecture (RXPA)**, while many standard library functions (like those in Classic REXX) are actually implemented in CREXX itself.

Libraries are housed in the `lib/` directory, which is divided into domains like:

- `lib/rxfnsb/` (Classic REXX Built-In Functions for Level B)
- `lib/rxfnsg/` (Level G class-shaped general-purpose interfaces)
- `lib/rxfnsl/` (Level L language-engineering examples and generated-output proving slices)
- `lib/rxfnsc/` (shared Level C/RexxScript runtime foundation, housing the REXX value, stem, and variable-pool classes plus directly callable Classic BIF implementations; `rexxclassicbif_call` is a deprecated compatibility path for current compiler output, while name-based intrinsic dispatch belongs to RexxScript)
- `lib/plugins/float/` (native scalar binary-float mathematics)
- `lib/plugins/stats/` (packed-float native statistics)
- `lib/plugins/vector/` (exact packed-vector computation and explicit portable float32 conversion)
- `lib/plugins/hash/`, `id/`, `fs/`, and `platform/` (narrow declarative native providers replacing the historical mixed `rxmath` and broad `system` bundles)

- `lib/rxfnsg/rexx/integer.crexx` and `decimal.crexx` (Level-G standard integer and decimal mathematics authored in Level B)
- `lib/plugins/` (General-purpose extensions like `fileio`, `regex`, `strings`, `socket`, etc.)
- `lib/ui/` (experimental backend-neutral event/effect/view contracts, a portable TUI driver, and an optional GTK driver)

Same-named Level B and Level C functions are separate APIs. For example, `lib/rxfnsb/rexx/value.crexx` is a read-only, immediate-caller metadata helper, whereas `lib/rxfnsc/RexxClassicBifValue.crexx` implements the Classic old-value/optional-assignment contract over `RexxValue` and `RexxVariablePool`. Direct Level C harnesses call selector functions without `rexxclassicbif_call`; compiler lowering to those entries is deferred to the later bulk lowering change.

The CMake build treats each `rxfnsc` member as an isolated source compilation. Its own source and only its declared transitive `rxfnsc` source dependencies are copied into a member-private, dependency-keyed source root under `lib/rxfnsc/members/`. External `library.rxbin`, `classlib.rxbin`, and `rxcexits.rxbin` imports are copied into the separate curated `lib/rxfnsc/imports/` root, and `rxc` runs with `--no-exe-import`. This gives each namespace one intended source or RXBIN provider while the library is being built; neither stale member files nor an older public `rxfnsc.rxbin` are eligible through the compiler executable directory. All members may build in parallel from the declared source dependency graph. `rxlink` writes the consolidated image under `lib/rxfnsc/linked/`, after which one publication action replaces `bin/rxfnsc.rxbin` through a temporary-file-and-rename step. Member actions never delete or rewrite another member's output or the public image.

The coherent source route is deliberate. The previous shared work directory could resolve some internal dependencies from source and others from generated RXBIN metadata depending on what was already present. Source/RXAS/RXBIN route parity is a separate compiler and metadata qualification obligation; do not restore mixed candidate visibility merely to reproduce an old linked-image hash.

`lib/rxfnsb/rexx/binary.crexx` provides the Level B binary helper surface. It contains the older Classic-style, 1-based, copy-returning helpers such as `binlength`, `binbyte`, `binsubstr`, `binoverlay`, `bininsert`, `bindelstr`, `binpos`, `bincompare`, `bin2x`, and `x2bin`. It also contains the Release 1 packed-memory helper surface, which is zero-based and mutates the first binary argument through `arg expose`: `binresize`, `binclear`, `binfill`, `binfillat`, `bincopy`, `binmemmove`, `binappend`, `binupdate`, `binmakegap`, and `bindrop`. These helpers are public fallbacks; compiler or inliner recognition for direct RXAS lowering should be added only for measured hot paths.

RexxScript is a first-class runtime product under `rexxscript/`, not a Level B BIF hidden inside `lib/rxfnsb`. It builds `bin/rexxscript.rxbin`, exposes the `rexxscript` namespace (`rexxscript_evaluate`, `rexxscript_evaluate_exposed`, `rexxscript_output`, `rexxscript_value`, and `.rexxscriptevaluator`), and carries the compatibility `rxfnsb.evaluate` facade for callers that still use the original prototype API. The `crexx` driver includes `rexxscript.rxbin` in its default runtime set; direct VM runs using the `REXXSCRIPT` compiler exit or compatibility `evaluate()` surface must load `rexxscript.rxbin` at runtime in addition to `library.rxbin`.

The same source directory also builds the standalone `bin/rexxscript` executable, which packages a file runner around an isolated `.rexxscriptevaluator()` instance.

The CMake build gives each RexxScript source member a private work directory under `rexxscript/members/`. Each primary source is copied to an isolated source root there, so sibling RexxScript sources cannot take precedence over the member RXBIN dependencies declared by CMake. Compiler imports are copied into the curated `rexxscript/imports/` staging directory, and `rxc` is invoked with `--no-exe-import` so an older executable-directory `rexxscript.rxbin` cannot be selected while its replacement is being built. The members are linked to the private `rexxscript/linked/rexxscript.rxbin` image. One publication action then replaces `bin/rexxscript.rxbin`; member actions do not delete or rewrite that shared product. The standalone runner has its own isolated source and curated import root containing only the four linked libraries it packages. This separation is a build-graph ownership rule: generated RXAS/RXBIN metadata remains owned by the action that generated it until the consolidated image is published.

The product master documentation is in `rexxscript/doc/`.

`lib/ui` contains the experimental Level G UI tracer architecture. Its public `CREXX` library, drivers, applications, launchers, and tests explicitly use `OPTIONS LEVELG`; the native GTK/RXPA C plugin remains lower-level mechanism. `ui.rxbin` owns logical views; `ui_contract.rxbin` owns typed messages, effect collections, component/driver interfaces and owner-driven sessions. The line, ANSI and optional GTK hosts all implement uidriver.run(session) directly. Each supplies capabilities(); there is no scalar adapter. With ENABLE_GTK=ON, ui_gtk.rxbin and rx_ui_gtk_native provide GTK. Nodes use stable IDs and relative root/below/right placement. RXPP's single UI_LAUNCHER composes every host with the same session factory. GTK signals enter an owning idle mailbox; dialog responses are deferred, and close intent is distinct from destruction. RXPA calls remain synchronous: the callback receiver holds a weak reference to the live host only for the duration of native run. See lib/ui/README.md, lib/ui/EXTENDING.md and examples/ui/text-inspector/README.md for the lifecycle and executable trail.

The first full-screen terminal host is ui_ansi.rxbin, using ui_contract directly. ui_dialogs, ui_terminal_view and ui_local_resources are Level G modules for standard confirmation/file selection, cell projection and scoped file grants. lib/plugins/console supplies the independent Level B rxconsole C provider: explicit console lease, timed key/text/paste input, resize, mouse, batched output and restoration. It has no ncurses or GTK dependency. RXPP's UI_LAUNCHER composes the same Text Inspector feature with this host. See lib/ui/TERMINAL.md for the lifecycle, precise limits and platform QA boundary.

lib/rxfnsb/rexx/rxjson.crexx contains the first JSON foundation library module for Level B web-service and transport work. It is implemented in Rexx, ships in library.rxbin, and intentionally exposes string-oriented helpers first:

- jsonvalid(json)
- jsontype(json, path)
- jsonget(json, path)
- jsoncount(json, path)
- jsonmembers(json, path, names[])
- jsonquote(text)
- jsonunquote(json)
- jsonarray(values[])
- jsonobject(keys[], values[])

Paths are Rexx-friendly and one-based for arrays, for example choices.1.message.content. This is enough to build LLM-style request JSON and extract common response fields without introducing a full object mapper yet. The parser internals use the binary-memory surface for the JSON source scan: the input is converted to .binary once per public helper call, structural bytes are read with direct <at..u8>

lowering, and unescaped object keys are matched with binary compare before any string materialization. Repeated extraction from the same large document still reparses per helper call; a parsed or indexed JSON handle remains the next product-level performance question.

For the full API contract, path syntax, examples, limits, and test location, see `lib/rxfnsb/rexx/rxjson.md`.

`lib/rxfnsb/rexx/rxsocket.crexx` provides the Level B wrapper for the VM's core TCP socket instructions. It ships in `library.rxbin` and exposes a small raw socket API for loopback clients, servers, and future web-service work:

- `socketcreate()`, `socketclose(sock)`
- `socketconnect(sock, host, port)`
- `socketconnecttls(sock, host, port)`
- `socketbind(sock, host, port)`, `socketlisten(sock, backlog)`, `socketaccept(sock)`
- `socketsend(sock, text)`, `socketrecv(sock, maxbytes)`
- `socketsendb(sock, data)`, `socketrecvb(sock, maxbytes)`
- `sockettimeout(sock, milliseconds)`, `socketblocking(sock, enabled)`, `socketnodelay(sock enabled)`, `socketkeepalive(sock, enabled)`
- `socketpending(sock)`, `socketshutdown(sock, how)`
- `socketpeer(sock)`, `socketlocal(sock)`, `socketstatus(sock)`, `socketerror(sock)`

The default library path is raw TCP and does not attempt HTTP parsing. Its core value is deployment stability: VM opcodes use platform socket APIs directly, avoiding dynamic `.rxplugin` discovery. Optional client TLS is exposed through the same VM-managed handle model as `socketconnecttls(sock, host, port)`, which connects and starts TLS before application bytes are exchanged. The instruction exists in all builds; without a TLS backend it returns a negative socket status rather than signalling. True STARTTLS remains a lower-level RXAS/VM instruction for future protocol-specific libraries and is not exposed by the public Level B `rxsocket` wrapper. Fresh CMake configurations select a TLS backend by platform: `NETWORK` on Apple platforms, `OPENSSL` on non-Windows Unix-like platforms, and `SCHANNEL` on Windows. `NETWORK` uses macOS Network.framework, Security.framework, CoreFoundation.framework, and the system trust store, `SCHANNEL` uses Windows SChannel/SSPI and the Windows trust store, and `OPENSSL` uses OpenSSL with default verification paths and host-name checks. `CREXX_ENABLE_TLS=OFF` can be used for dependency-minimal builds.

`CREXX_TLS_STATIC_OPENSSL=ON` asks CMake to prefer static OpenSSL libraries when the OpenSSL backend is selected. For API details, status codes, and examples, see `lib/rxfnsb/rexx/rxsocket.md`.

The older OpenSSL-backed dynamic socket plugin is deprecated and no longer builds by default. Developers who still need it can configure with `CREXX_BUILD_LEGACY_SOCKET_PLUGIN=ON`; otherwise source builds avoid that plugin's OpenSSL discovery and distribution burden.

The Level G CMake build does not scan the shared product `bin/` directory. Each module compiles from a source root containing only its primary source, with `library.rxbin` and `classlib.rxbin` copied to a curated base import root. The HTTP members receive separate roots containing only their declared private HTTP prerequisites; the current public HTTP member is staged as `rxfnsg.rxbin` for the same-namespace server and LLM consumers. Every compile uses `--no-exe-import`, so an older consolidated `rxfnsg.rxbin`, a sibling source, or an unrelated plugin cannot supply build-time metadata. The Level L build follows the same rule with its narrower `library.rxbin`-only import root.

`lib/rxfnsg/rexx/httpcore.crexx` is the private Level B `_rxhttpcore` backend for binary HTTP framing, parsing and codecs. It is shared by the Level G client, server and LLM providers and is not a public convenience client. The public pre-release HTTP surface is intentionally Level G so that policy, bounded task lifecycle and typed request/response values are not duplicated at two language levels.

`lib/rxfnsb/rexx/trace.crexx` provides the Level B trace/debugger internals used by `rxdb` and by the `TRACE` certified compiler exit:

- `.tracecontroller`: breakpoint enable/disable, module/procedure helpers, source/ASM lookup, default runtime/debugger filtering, and shared structured trace-event metadata lookup
- `.tracecontext`: immutable per-event module/address/source/ASM/procedure snapshot
- `.trace_interrupt_raw`: internal register-mapped view of the VM interrupt object used by breakpoint handlers
- `_trace_set(mode)`, `_trace_set_from_env(allow_output)`, `_trace_set_format(format)`, `_trace_set_output(target)`, `_trace_current_mode()`, `_trace_mode_from_option(option)`, `_trace_needs_breakpoints(mode)`, `_trace_context_from_raw(raw)`, `_trace_should_trace_context(event)`, `_trace_should_emit_key(key)`,

_trace_capture_result_target(module, addr), _trace_pending_parent_-register(module, type, value), _trace_supply_parent_value(value), _-trace_pending_result(), _trace_pending_prefix(), and _trace_clear_-pending_result(): the compiler-exit-facing runtime surface for setting trace mode/format/output, normalizing dynamic TRACE VALUE options, applying explicit TRACE ENV CREXX_TRACE / CREXX_TRACE_TO environment settings, coordinating simple TRACE R/TRACE I parent-frame value reads, and servicing BREAKPOINT events. .tracecontroller owns the trace-event metadata lookup and pending value state so other trace/debug users can share the same interpretation. The exit still emits caller-frame assembler to enable/disable breakpoints, install the handler, and perform the actual metalinkpreg read for a pending register, because VM signal tables and metalinkpreg are frame-sensitive. Register reads are driven only by .traceevent metadata; .meta_reg remains scope/register-placement metadata, not a value-change stream.

- _trace_command_before(environment, command) and _trace_command_after(environment, command, rc, condition): ADDRESS dispatch hooks used by TRACE C, TRACE E, TRACE F, and quiet/default TRACE N.

Classic text and LLM trace formatting belong to the generated TRACE exit handler. The shared runtime provides controller/filter helpers and output plumbing; RXDB and other debugger UIs should make their own presentation and stepping choices from the structured metadata. _trace_set_output accepts stdout, stderr, or a file path; file targets are opened in append mode per trace record.

The helpers rely on VM metadata instructions such as metaloaddata, metaloadinst, metadecodeinst, and metaloadedmodules, so deployable linked images that strip source/TRACE debug metadata may still provide ASM/module data while source-line and trace-event lookup return empty. Debugger UI text and menu rendering belong to debugger/rxdb_gui.crexx, not the library trace internals.

Trace contexts cache their metadata. Breakpoint events resolve procedure data for namespace filtering first, then load source/instruction content only after the event is accepted. Namespace matching operates on already-normalized stored components, and trace escaping scans/appends code points directly. Invalid direct _trace_set modes raise INVALID_ARGUMENTS; failure to open a configured output target raises NOTREADY. VM metadata/module queries retain their zero/empty status protocols.

`lib/rxfnsb/rexx/_address.crexx` owns the Rexx-side ADDRESS protocol. In addition to command dispatch, redirects, sandboxes, and function calls, `addressrequest.get_binding_value(name)` gives REXX providers a simple way to read scalar bindings auto-exposed from ADDRESS host-variable anchors such as `:name` and `${name}`. Anchor interpretation remains provider-specific; the VM only carries binding values and write-back updates.

`lib/rxfnsg/rexx/llm.crexx` contains the Level G LLM integration surface. The module is `options levelg`, builds into `rxfnsg.rxbin`, uses `rxjson`, and sends through the public Level G `.httpclient` plus the shared private `_rxhttpcore`. It exposes a class-shaped interface in the `rxfnsg` namespace:

- `llm`: provider-selecting interface for the local Ollama default
- `ollama`: concrete local Ollama implementation over plain HTTP
- `openai`: concrete OpenAI Responses API implementation over HTTPS
- `anthropic`: concrete Anthropic Messages API implementation over HTTPS
- `gemini`: concrete Gemini `generateContent` implementation over HTTPS

The first provider posts JSON to a local Ollama `/api/generate` endpoint with `stream:false`. It keeps reconstructed HTTP diagnostics plus the decoded JSON body available. Hosted providers are also REXX implementations: they use environment-variable API keys by default, build provider-specific JSON request bodies and authentication headers, and send through the same pooled client with TLS enabled. Each provider closes its owned HTTP pool through `close()`. `demos/llm/llm_address_environment.crexx` layers a REXX ADDRESS provider over these clients so scripts can use model-shaped environments such as `ADDRESS LLM_GPT_4_1`, `ADDRESS CLAUDE_SONNET_4_5`, and `ADDRESS GEMMA4_LATEST`; the provider caches by environment instance and routes through a small driver registry of exact aliases and prefixes. See `lib/rxfnsg/rexx/llm.md` and `demos/llm/`.

`lib/rxfnsg/rexx/http.crexx` and `httpserver.crexx` are the initial Level G HTTP surface. The transferable `.httpclient.pooled(origin, connections, admission, maximum_response, ?policy)` supports buffered `request`, `get` and `post` task methods plus endpoint-backed `post_stream`. It exposes `.httpheaders`, `.httppolicy`, `.httpbody` and independent typed `.httpresponse` values. Each long-lived `.taskwork` connection owner holds at most one reusable socket. Fixed-size admission descriptors and type-4 byte endpoints carry canonical references and bytes, never socket integers or live VM values. The client provides safe headers, bounded budgets, verified TLS, same-origin 307/308 and idempotency-key replay policy, ambiguity history, fixed/chunked request streams, identity-encoded response streams and bounded gzip/zlib/raw-DEFLATE decoding for buffered responses.

The clear-text buffered `.httpserver` owns every accepted socket on its controller and sends only complete `.httprequest` records to sealed `.httpservice .taskwork` targets. Handlers return `.httpresponse.text(...)` or `.httpresponse.binary(...)`; the controller revalidates the result against its limits. This does not use or implement `.taskscope.ask()` or `.serviceref`. Server TLS, HTTP/2, WebSockets, background lifecycle and streaming handlers are outside the current contract. See the concurrent HTTP client/server guide and the both-VM `lib/rxfnsg/tests_functional/ts_http_*` fixtures.

`lib/rxfnsl/rexx/tinyexpr.crexx` contains the first Level L language-engineering proving slice. It is deliberately not a lexer generator or parser generator yet. Instead, it is hand-written in the shape that a future generator might emit: a packed binary character-class table, fixed-width binary token records, an exposed declaration procedure for generated token/layout constants, direct binary-memory reads/writes, a zero-copy lexeme compare helper, and a tiny precedence parser over the packed token stream. The purpose is to learn whether the Rexx/RXAS binary-memory surface is usable for generated language tooling before porting a tool such as re2c or changing a generator backend to emit this style directly. See `lib/rxfnsl/rexx/tinyexpr.md`.

A.1 1. BIFs Implemented in CREXX (`lib/rxfnsb/rexx/`)

A significant portion of the Classic REXX Built-In Functions (such as `abs()`, `date()`, `length()`, `substr()`) are written entirely in cREXX. These are located in `lib/rxfnsb/rexx/`.

This approach minimizes the VM footprint and demonstrates the capability of the CREXX compiler to handle system-level logic.

For repo-native Level B authoring patterns, argument signature examples, and wayfinding to real `.crexx` examples, see `docs/ai-context/CREXX_LEVELB_AUTHORING.md`.

`lib/rxfnsb/rexx/fileio.crexx` exposes the sequential Level B text file BIFs `linein`, `lineout`, `charin`, `charout`, and `lines`. These are UTF text functions over

`.string`: `linein` reads one line without its line terminator, `lineout` writes text plus a newline, `charin` reads UTF-8 codepoints, and `charout` writes text without adding a newline. They are not the binary byte I/O surface. On `stdin`, `linein` returns as soon as the line terminator is read; it does not probe for a following byte. A trailing line terminator at physical EOF does not create a synthetic empty `linein` record; physical blank lines are still preserved. `lines(name)` returns `-1` when the named stream cannot be opened so callers can distinguish missing/unreadable input from an empty file.

The same module exposes the deliberately small whole-file binary surface `readbinary(path) -> .binary` and `writebinary(path, data = .binary) -> .int`. `readbinary` accumulates exact bytes in 32 KiB chunks; `writebinary` replaces the target and returns the byte count. Both use binary file modes and the VM `freadb/fwriteb` path, so embedded NUL and bytes that are not valid UTF-8 are preserved without newline translation. I/O failure raises `NOTREADY`. These are whole-value operations, not an incremental `rxio` stream abstraction, so callers remain responsible for choosing files that sensibly fit in memory.

A.1.1 Build And Debugging Rules

The Rexx BIF library build is a bootstrap build. `lib/rxfnsb/rexx/CMakeLists.txt` compiles most Rexx BIF modules with `rxc -x --import-rxas`, which disables certified compiler exits while building the library used by those exits. An explicit `TRACE`, `PARSE`, `ADDRESS`, or other certified-exit statement added directly to a BIF source file will fail with `#CERTIFIED_EXIT_DISABLED`.

Do not debug BIFs by adding `TRACE RESULTS` inside `lib/rxfnsb/rexx/abs.crexx` or another library source file. Use one of these instead:

- a normal scratch program or functional test that imports `rxfnsb` and calls the BIF with compiler exits enabled;
- `TRACE UNSUPPRESS NAMESPACE rxfnsb` (or `rxfnsg`, `rxfnsl`, `rxfnsc`) when library frames should be visible despite the default system-namespace filter;
- `TRACE ASM` or `TRACE LLM` from the caller for lower-level metadata checks;
- `crexx -native --link-keep-source` or an unstripped `rxlink` image when debugging native/linked output, because stripped linked images drop `META_SOURCE_STEP` and `META_TRACE_EVENT`.

Useful focused checks for this area:

```
cmake --build cmake-build-debug --target testbifs
ctest --test-dir cmake-build-debug -R '^ts_.*_(noopt|opt)$' --output-on
    -failure
ctest --test-dir cmake-build-debug -R '^test_system$' --output-on-
    failure
ctest --test-dir cmake-build-debug \
  -R '^(trace_event_metadata|test_trace_|ts_trace_|rxlink_format_check|
     rxlink_rxdas_strip_smoke)' \
  --output-on-failure
```

The standard Level B array helpers live in `lib/rxfnsb/rexx` and are preferred over the legacy `lib/plugins/arrays` RXPA plugin. Current array BIFs include `arrayfind`, `arrayinsert`, `arraydelete`, `arraysort`, `arraycopy`, `arraydrop`, `arrayhi`, `arraymove`, `arrayappend`, `arrayprepend`, `arraypop`, `arrayshift`, `arrayget`, `arrayset`, `arraycontains`, `arrayindexof`, `arrayreverse`, and `arrayjoin`. The former diagnostic `arraydump` and `arrayformat` procedures are deployed as the `arrayformatdemo` module under `examples/functions/array-formatting`; they are demonstration support rather than Level B or Level G library contracts. These `array*` helpers are currently the `.string[]` helper family, not generic or numeric typed-array helpers. For `.object[]`, use the separate `objectarray*` family below. For raw one-dimensional dynamic typed arrays, including numeric arrays, Level B now has core `rxc` statement forms: `append array with value`, `insert array with value at index`, `remove array at index [for count]`, `remove array at first to last`, and `clear array`. These are compiler syntax, not exits or public helper BIFs, and lower directly to VM array attribute opcodes. Object-shaped mutating helpers are exposed separately as `objectarrayinsert`, `objectarraydelete`, `objectarrayappend`, `objectarrayprepend`, and `objectarraydrop`, plus `objectarraymove` for block moves. Insertion, deletion, movement, append, prepend, pop, shift, gap-growing set, and the object-array insert/delete/append/prepend/drop/move helpers use VM bulk attribute instructions so the logical array pointer list can be shifted without a Rexx-level per-element copy loop. Mutating array BIFs must declare the array with `arg expose`. To clear an existing array object through the helper surface, use `arraydrop` for `.string[]` and `objectarraydrop` for `.object[]`; `clear array` is the source-level raw dynamic-array clear statement.

Container naming is intentionally split by shape:

- Arrays/lists are ordered, one-based, duplicate-preserving sequences using array[0] as the high water mark. Classic code should use the array* BIFs; OO code should expose the same semantics through StringArrayList/StringLinkedList methods such as add/append, insert, remove, get, set, size, clear, contains, and indexOf. The Release 1 Rexx-first iterator direction is explicit: iterator() returns an unsynchronized live iterator over the current collection, while snapshotIterator() returns an iterator over a factory-time snapshot.

- Maps/stems are keyed containers. Do not overload array* for keyed lookup; use stem* or map* BIF names and class methods such as put, get, containsKey, remove, keys, and values. The current pure-Rexx classlib map iterators are factory-time snapshots built from key/value arrays. Do not infer StringArrayList.iterator() live-reference semantics for maps.
- Sets are uniqueness containers. Use add, contains, remove, size, clear, and toArray/fromArray semantics consistently across classic and class-shaped APIs. The current pure-Rexx classlib set iterators are also snapshot iterators.

lib/rxfnsb/rexx/stem.crexx is the Level B keyed-container implementation. Its public hash method retains the conventional Rexx-facing polynomial hash, but factory/get/set/default-reset/size/key/value methods use the NR-15 native stem instructions. The private D2-hybrid runtime layout keeps hash metadata in the receiver binary and strings in ordinary VM values; callers must not inspect or persist those bytes. The compiler may lower exact calls on a proved simple concrete rxfnsb.stem receiver directly. Class attributes, computed receivers, and other shapes keep normal call/copyback behavior unless their storage proof is equally strong. Multi-tail source expressions retain canonical construction where conversion, evaluation order, or TRACE observation has not proved the two-segment form equivalent.

lib/classlib/Concurrency.crexx is the explicit Level B concurrency surface. It ships in classlib.rxbin and implements the pool, scope, task, target/work/context, completion, channel/request, ChannelValue/codec, byte-endpoint, service-reference and transfer-buffer interfaces. Local and isolated-process pool paths reach RXVM only through authored chanopen, chanstart, chanwait, chancancel and chanclose instructions; there is no RXPA task-start path or procedure-name-string dispatch. Level G task procedures/methods, task expressions and DO PARALLEL lower through these classes, including receiver-side .taskwork factories. The syntax is gated by OPTIONS LEVELG; Level B programs may use the explicit classes directly. Service ask and pool statistics deliberately signal unsupported status 19.

`.taskcontext.endpoint()` reconstructs a worker-local byte endpoint from a transferable type-4 provider reference and has a direct public-contract test through the full toolchain and both VM variants. Functional tests are in `lib/classlib/tests_functional/testConcurrency.crexx`; `testTaskContextEndpoint.crexx` and imported task method/`.taskwork` tests exercise `rxc`, `rxas`, `rxlink` and both VMs. The enduring architecture and status boundary is `CREXX_CONCURRENCY.md`.

Level B classlib collection names carry their value contract because the language does not yet have generics. Current public classlib containers and iterator interfaces are therefore explicitly tagged:

- `String...` classes store string values, for example `StringIterator`, `StringIterable`, `StringArrayList`, `StringHashMap`, `StringTreeMap`, `StringStack`, and related iterator/set/list classes.
- `Object...` classes store object values where no key contract is involved, for example `ObjectIterator`, `ObjectIterable`, `ObjectArrayList`, `ObjectLinkedList`, and `ObjectStack`. Callers explicitly upcast concrete class instances with `as .object` when storing them.
- `StringObject...` map classes use string keys and object values, for example `StringObjectHashMap` and `StringObjectTreeMap`. They store keys in REXX string arrays and values in REXX object arrays.

The current public classlib collection surfaces are Rexx-only. `StringHashMap`, `StringTreeMap`, `StringHashSet`, `StringTreeSet`, `StringLinkedList`, and the string-key/object-value map variants no longer require the historical native `treemap` or `llist` plugins. `StringTreeMap` is backed by an AVL node pool in parallel arrays; `StringOldTreeMap` retains the previous array-backed map only for comparative benchmarks. Temporary `StringTreeMapV2/V2A/V3/V4` experiments were used to measure binary-memory and source-shape alternatives, then removed because none improved on the production `StringTreeMap` overall. String-key lookup uses strict equality so empty string keys and blank string keys remain distinct.

The core `classlib.rxbin` CMake build gives each of its 53 members a private work directory under `lib/classlib/members/`. External `library.rxbin` and `rxcexits.rxbin` inputs are copied into a curated import root, and every `rxc` invocation uses `--no-exe-import`. Internal providers have an explicit route: `CLASS_DEPS_*` entries are private generated RXBIN inputs, while `CLASS_SOURCE_DEPS_*` entries are copied into the member's dependency-keyed source root. The split is required because some current class cycles and interface signatures remain source/RXBIN-route-sensitive. A member sees only its declared providers; an unrelated member that happens to finish first and an older public `classlib.rxbin` are never candidates.

After all member actions complete, `rxlink` writes the core image under `lib/classlib/linked/main/`. One publication action then replaces `bin/classlib.rxbin` through a temporary-file-and-rename step. Member actions do not delete or rewrite the public image or another member's output. Keep the source and RXBIN dependency tables separate until permanent route-parity tests prove that a single route is semantically interchangeable.

`Id`, `KeyDB`, and `Os` are intentionally kept out of the core `classlib.rxbin` image so products such as RexxScript can use the class library without pulling in unrelated native plugins. They are built and tested as the opt-in `classlib_native.rxbin` adapter image and depend on the `id`, `keyaccess`, and `system` plugins respectively. Build, test, and package changes that expose these classes must include their native plugin runtime modules. Do not rewrite these wrappers in Rexx merely to avoid native code; classify or remove the underlying capability explicitly if it is not part of the intended release surface.

The native adapter build uses the same ownership discipline as the core classlib. `library.rxbin` and `rxcexits.rxbin` are staged in a curated base import root, while every adapter receives a dependency-keyed private source and plugin root. `Id` sees only `rxid.rxplugin`, `KeyDB` sees only `rx_keyaccess.rxplugin`, and `Os` sees only `rxfs.rxplugin` and `rxplatform.rxplugin`. Each member runs `rxc --no-exe-import` and `rxas` in its own directory. `rxlink` writes the complete image under `lib/classlib/linked/native/`, and one temporary-file-and-rename action publishes `bin/classlib_native.rxbin`. Do not add a plugin to a broad shared search directory; declare it against the adapter that imports it.

Bare collection names such as `Iterator`, `Iterable`, `ArrayList`, `HashMap`, `TreeMap`, and `Stack` are intentionally left free for future Level G generic or generic-like surfaces. Object-key maps and object sets are also deferred until the language has a clear object equality/hash/ordering contract.

Imported Rexx BIF calls inline only when the imported artifact carries `META_INLINE`. `rxlink` strips `META_INLINE` by default, so the standard-library link explicitly uses `PRESERVE INLINE`. Release linked libraries also use `STRIP SOURCE`, keeping inline bodies available to downstream optimisation while dropping source-level debug metadata: `META_SOURCE_STEP` file/source-line records and `META_TRACE_-EVENT` semantic TRACE value records. Debug linked libraries keep both inline bodies and source/TRACE metadata. Class-library hot paths that rely on this should inspect generated `.rxas` when changing import paths or link-strip policy.

RXAS review matters for performance-sensitive Level B classlib code. The `StringTreeMap` AVL rewrite showed that an inlined private lookup helper still left block-expression scaffolding in generated RXAS; writing `get()` and `containsKey()` as direct loops cut Release lookup time for 2,500 entries from about 200 ms to about 2.3 ms on the local arm64 development machine. Keep hot lookup/update loops direct unless RXAS inspection proves the helper shape is equivalent.

The binary-backed treemap trials showed that packed metadata can be expressed cleanly with `.binary` and direct binary-memory lowering. Source-shape caching helps both the array/register and binary versions; a packed `.u32`/`.u8` binary layout improves the binary variant versus the first 64-bit-field cut. The current optimized `.int[]` AVL metadata remains the best overall shape for this workload. The measurements are retained in `docs/planning/beta-3/notes/string-avl-treemap-trial.md` as evidence for binary-surface ergonomics and missing intrinsics, not as live classlib APIs.

The focused `binary_fastpath_compare` benchmark isolates scalar binary-memory instructions from collection algorithms. In Release builds, direct `.u8`, `.u32`, and `.int` binary reads/writes are substantially faster than indexed `.int[]` attribute-array access. A VM fast-path change keeps strict bounds checks and canonical little-endian semantics, but replaces byte-by-byte fixed-width load/store loops with `memcpy`-based 1/2/4/8-byte helpers plus byte-swap only on known big-endian hosts. A temporary unsafe no-upper-bound experiment showed only modest gains, so Release 1 should keep strict checked binary access. See `docs/planning/beta-3/notes/bina` for timings and user guidance.

`lib/plugins/arrays` is deprecated and retained only as a legacy plugin smoke test. New Level B code should import `rxfnsb` and use the standard BIFs.

A.1.2 Anatomy of a cREXX BIF

Functions written in cREXX follow standard language rules, utilizing namespaces and type enforcement:

```
/* lib/rxfnsb/rexx/abs.crexx */
options levelb

namespace rxfnsb expose abs

abs: procedure = .string
  arg number = .string
  if left(number, 1) = '-' then number = substr(number, 2)
  return number
```

Key Features:

1. **Namespaces:** Functions must declare `namespace rxfnsb expose <function_-name>` so they correctly bind into the Standard Library space that user scripts import.
2. **Inline Assembly (`assembler`):** When low-level access is required (such as fetching the current system time in `date.crexx`), cREXX BIFs can drop down into inline bytecode using the `assembler` keyword.
3. **Compilation:** These `.crexx` files are compiled into `.rxbin` bytecodes during the build process and are packaged or shipped exactly like user-compiled binaries.
4. **Explicit Late Load:** `loadmodule(path) -> .int` wraps the VM's `METALOADMODULE` instruction. Use it when Rexx code deliberately loads a `.rxbin` or `.rxplugin` provider before calling imports supplied by that file. The VM relinks immediately after a successful load.
5. **ADDRESS Public Helpers:** `addressenv(name) -> .addressenvironment` returns the cached environment object, including `environment_name()` and `environment_id()` for the traditional `SYSTEM`/`PATH` environments and for Rexx/native providers. `addresscall(env, name, ...) -> .string` wraps the lower-level `_address_function(env, name, args[])` request object path. `_address_call(...)` and `_address_call_response(...)` remain internal compatibility spellings for code that needs the raw response object.

A.2 2. RXPA (cREXX Plugin Architecture)

For functions requiring native performance or access to C-level system libraries (like cryptography or sockets), `crexx` provides the RXPA framework. This macro-driven C API (defined in `rxpa/crexxpa.h` and `rxpa/rxpa.h`) allows developers to write REXX-callable functions without interacting directly with the VM's internal `stack_frame` or `value` structures.

Plugins can be compiled in two ways:

1. **Dynamic Plugins (`.rxplugin`)**: Recommended for user extensions. The compiler discovers declarations on binary import roots. At runtime an explicitly loaded legacy plugin still uses module search paths, while a compiled native dependency is resolved declaratively through a trusted `<provider-id>.rxplugin` artifact.
2. **Static Plugins**: Built directly into the `crexx` binaries. These are typically reserved for core Standard Libraries to guarantee they are always available.

A.2.1 RXPA concurrency contract

An unmodified RXPA plugin remains valid in a multi-VM process. The host treats it as a **legacy process-shared plugin**. While the process has only one VM that has loaded a legacy plugin, its procedures use the same direct adapter as the single-threaded product. Loading a legacy plugin into a second VM starts one cold, sticky transition: the host waits for existing direct legacy execution to leave its VM execution boundary, rebinds every live legacy procedure to one process-wide recursive compatibility lock, and publishes the new load only after rebinding. A VM that loads only process-reentrant plugins does not trigger the transition. Once concurrent legacy mode has been entered, later legacy loads remain locked for the rest of the process lifetime.

This is conservative because the existing initializer ABI has no way to prove that the plugin's C statics, its dependencies, or its error paths tolerate concurrent entry. Recursive locking allows a legacy plugin to make a nested call that reaches another legacy RXPA procedure without deadlocking. The cold transition can wait for a long-running legacy-capable VM invocation to return; plugin load must not assume that publication is instantaneous.

An audited plugin that is safe for concurrent entry can opt in by adding one file-scope declaration after including `crexxpa.h`:

```
#include "crexxpa.h"

RXPA_PLUGIN_PROCESS_REENTRANT
```

The declaration is plugin-wide and must appear exactly once in a dynamic plugin. It exports an optional versioned manifest; it does not change `_initfuncs(rxpa_initctxptr)`, `rxpa_libfunc`, `ADDPROC`, the language-level option string, RXAS, or RXBIN. Older hosts ignore the extra symbol. New hosts treat an absent, malformed, unsupported, or unknown manifest as legacy mode. Static plugins use the same source declaration and are associated with their rebuild-together `PLUGIN_ID`.

`PROCESS_REENTRANT` means that all procedures published by that plugin may be entered concurrently and still have defined behavior. It does **not** mean side-effect-free. Synchronized logging or I/O, atomics, and calls into documented thread-safe services are fine. Before adding the macro, audit:

- every writable file-scope/static variable;
- lazy initialization, caches, random-number state, locale and error buffers;
- all libraries and OS APIs called by every published procedure;
- cleanup and failure paths, including calls made during nested execution; and
- any assumption that one call completes before the next starts.

A function that uses only stack locals, immutable tables and RXPA helpers is a typical candidate. A plugin that increments an ordinary static counter, reuses one static work buffer, changes process locale, or relies on an undocumented non-thread-safe library is not; leave it unmarked or add its own synchronization first. Plugin maintainers should make the assertion wherever this audit passes: the binding then remains permanently direct, regardless of how many other VMs or OS threads the process starts.

The P1 capability applies to procedure calls. Native-payload `copy` and `finalize` callbacks remain on the recursive compatibility lane because they can run outside the originating procedure and need a separate lifetime claim.

Plugins with mixed policy can instead export the V2 procedure query:

```
static uint32_t procedure_capabilities(const char *name) {
    if (strcmp(name, "example.stateless") == 0) {
        return RXPA_PROCEDURE_CAP_PROCESS_REENTRANT;
```

```
    }
    return 0u; /* conservative legacy lane */
}

RXPA_PLUGIN_PROCEDURE_CAPABILITIES(procedure_capabilities)
```

The query runs while the plugin is loaded and returns exactly one of `0`, `RXPA_PROCEDURE_CAP_PROCESS_REENTRANT`, or `RXPA_PROCEDURE_CAP_SESSION_AFFINE`. Unknown bits, both known bits together, or a session-affine result without a complete session hook set fail closed to legacy behavior. The host stores the selected invoker in the procedure at load; ordinary calls do not repeat the query or branch on the capability.

A plugin whose mutable native resources belong to one VM uses `RXPA_PLUGIN_SESSION_AWARE(create, destroy, enter, leave, query)`. The host creates one session per VM/plugin load, rejects the load if creation fails, enters that session around each session-affine procedure call, and destroys it after VM values/modules are released but before the plugin DSO closes. `enter` must store the previous thread-local session in its output cookie and `leave` must restore it, so nested native calls are safe. Session callbacks themselves must not retain VM-owned `value *` or register handles after the call.

The optional V2 symbol does not change `_initfuncs`, `rxpa_libfunc`, `ADDPROC`, RXAS, or RXBIN. Older hosts ignore it and call the unchanged procedures as a legacy plugin. A session-aware plugin that promises old-host compatibility must therefore provide its own process-default session when no V2 `enter` callback has selected a per-VM session; ODBC is the reference implementation.

Current bundled classification is deliberately conservative:

Classification	Bundled examples	Rule
Plugin-wide process-reentrant	`cipher`, `rx_hash`, `rxfloat`, `rxstats`, `rxvector`, `rxid`, `rxfs`, `rxplatform`, `stack`, `strings`, `getpi`	Audited/repaired and marked with `RXPA_PLUGIN_PROCESS_REENTRANT`. `rxfloat` also publishes direct `rxmath` scalar compatibility names; the historical `inlinec`, statistics, hash and UUID mixture and the broad `system` provider are removed.

Classification	Bundled examples	Rule
Per-VM session	`odbc`	Database procedures are session-affine; `odbc.show_message` is process-reentrant; old hosts use the plugin's default session.
Unqualified	All other bundled plugins	Remain legacy and serialized until their complete state, dependencies, failure paths and teardown have been audited.

ODBC testing has two layers. The test-only mock driver is always built when `BUILD_TESTING` is enabled and deterministically covers failure injection, retained parameter pointers, concurrent per-VM sessions, teardown and the old-host default session. When `ENABLE_ODBC=ON` and a `sqlite3odbc` library is discoverable, CMake also generates a private SQLite `:memory:` DSN and adds `odbc_sqlite_prepared_rxbvm` and `odbc_sqlite_prepared_rxtvm`. Those tests load the real unixODBC plugin/driver and cover prepared binds, independent active statements, reset/re-execute, fetch, transactions, metadata and diagnostics. Set the `CREXX_SQLITE_ODBC_DRIVER` CMake cache path explicitly when the driver is installed outside the standard Unix, `/usr/local`, or Homebrew locations.

A.3 3. Writing a Native Function

A native C function meant to be exposed to REXX is defined using the `PROCEDURE` macro.

A.3.1 Argument Access and Returns

The VM passes arguments as opaque handles mapped to internal VM registers. The RXPA headers provide macros to extract native C types from these registers and to write results back:

- `NUM_ARGS`: The count of arguments passed from REXX.
- `ARG(n)`: Retrieves the opaque handle for the *n*th argument.

- `GETINT()`, `GETFLOAT()`, `GETSTRING()`: Extracts the native C value from a register handle.
- `SETINT()`, `SETFLOAT()`, `SETSTRING()`: Writes a native C value into a target register.
  `SETSTRING()` copies the supplied NUL-terminated bytes into VM-owned value storage. The plugin retains ownership of the source buffer and must release it after `SETSTRING()` when that buffer was allocated by the plugin.
- `SETNATIVEPAYLOAD()` / `GETNATIVEPAYLOAD()`: Attach or read binary payload storage. This is the RXPA path for ordinary `.binary` arguments and results, as used by `rxhash.sha256()`. Object-shaped native payloads additionally require a clear copy/finalizer contract; Rexx code never sees a C pointer.
- `ISINITIALIZED()`: Non-raising test of the language-level typed-object initialization flag. Payload-consuming functions use this before treating a zero-length object payload as an initialized empty value. RXPA does not reject bare typed objects automatically because ordinary procedures may legitimately accept one in order to inspect its initialization state.
- `RETURN`: The specific target register designated for the function's return value.

A.3.2 Error Handling

Errors are thrown using the `RETURNSIGNAL` macro, which halts execution and raises a specific `RXSIGNAL_*` exception within the VM. Successful execution must conclude with `RESETSIGNAL`.

A.3.3 Example Native Function

```
#include "crexxpa.h"

// Example: Add two integers together
PROCEDURE(add_integers)
{
    int result;

    // 1. Validate argument count
    if (NUM_ARGS != 2) {
        RETURNSIGNAL(SIGNAL_INVALID_ARGUMENTS, "2 arguments expected")
    }

    // 2. Extract values and perform logic
```

```
    result = GETINT(ARG(0)) + GETINT(ARG(1));

    // 3. Set the return value
    SETINT(RETURN, result);

    // 4. Clear the signal state and exit
    RESETSIGNAL
}
```

A.4 4. Registering Functions with the VM

Once the C functions are written, they must be registered so the REXX compiler (`rxc`) and interpreter (`rxvm`) can map REXX namespace calls to the native C function pointers.

This is accomplished using the `LOADFUNCS` mapping block, which binds the C function pointer to a REXX namespace, declares its return type, and defines its expected argument signature.

```
// Publish functions to the cREXX compiler and VM
LOADFUNCS
//       C Function       REXX Namespace & Name      Opt  Return Type
      Argument Signature
ADDPROC( add_integers,    "rxexample.add_integers",  "b", ".int",
          "i1 = .int, i2 = .int" );
ADDPROC( string_concat,   "rxstr.string_concat",     "b", ".string",
       "s1 = .string, s2 = .string" );
ENDLOADFUNCS
```

A.4.1 Registration Breakdown:

1. **C Function**: The literal name of the C function defined by `PROCEDURE(...)`.
2. **REXX Namespace & Name**: How the function will be called from REXX code (e.g., `import rxexample; x = add_integers(1, 2)`).
3. **Option**: The target CREXX language level (`"b"` for Level B, `"c"` for Level C).
4. **Return Type**: A string literal dictating the exact CREXX type returned (`".int"`, `".string"`, `".void"`).
5. **Arguments**: The exact type signature expected by the compiler. It supports standard types, array syntax (`.int[]`), and reference passing (`expose`).

During compilation, `rxc` parses this block to strictly enforce type safety when the REXX code invokes the native plugin. It also retains the manifest's stable provider ID and emits `.provider` metadata for used native declarations. During execution, `rxvm` resolves that ID to an already linked static provider or a trusted `<provider-id>.rxplugin`, verifies the binary's provider identity and signature, and only then links the callable into the execution space. No REXX wrapper is required.

For a provider with both delivery forms, use the build helper after declaring the targets:

```
add_dynamic_plugin_target(_example example.c)
add_static_plugin_target(_example example.c)
add_rxpa_provider_package(_example)
```

It publishes `bin/providers/rx_example.rxplugin`, the canonical native archive `rx_example.a` (or `rx_example.lib`), and the compatibility archive `rx_example_-static.a` (or `.lib`). `crexx -native` reads the same RXBIN requirements via `rxlink -p`, prefers the canonical archive, falls back to the compatibility name, and retains its static registration automatically. Tests and application-local package builds may pass `OUTPUT_DIRECTORY`; omitting it deliberately selects the standard build/install provider directory.

A deliberate alternate implementation, such as a mock provider, may retain a distinct CMake target while publishing the same stable manifest identity:

```
add_dynamic_plugin_target(_example_mock PROVIDER_ID rx_example mock.c)
```

Its delivered artifact must still use the canonical `rx_example.rxplugin` stem. `PROVIDER_ID` is not an aliasing mechanism: the runtime requires the requested artifact stem and embedded manifest identity to agree.

A.4.2 Standard `rx_hash` provider

`rx_hash` is a B+G standard/default provider, not compiler or VM core. It is built and installed in dynamic and static forms and publishes one-shot, incremental, hexadecimal, and bounded-memory file SHA-256 plus four named 32-bit hash/checksum procedures:

```
import rxhash

digest = rxhash..sha256(data)
hex_digest = rxhash..sha256hex(data)
state = rxhash..sha256init()
state = rxhash..sha256update(state, data)
stream_digest = rxhash..sha256final(state)
```

```
file_digest = rxhash..sha256file(path)
table_hash = rxhash..djb2(data)
seeded_hash = rxhash..murmur3(data, seed)
fnv_hash = rxhash..fnv1a(data)
checksum = rxhash..crc32(data)
```

The exact SHA-256 family is `sha256(data = .binary) = .binary`, `sha256hex(data = .binary) = .string`, `sha256init() = .binary`, `sha256update(state = .binary, data = .binary) = .binary`, `sha256final(state = .binary) = .binary`, `sha256finalhex(state = .binary) = .string`, `sha256file(path = .string) = .binary`, and `sha256filehex(path = .string) = .string`, all in namespace `rxhash`. Raw results are exactly 32 bytes and direct hex results are exactly 64 canonical lowercase characters. Embedded zero and invalid UTF-8 bytes are data.

Incremental state is a 152-byte versioned, big-endian, pointer-free binary value containing a canonical header, byte count, chaining words, pending-byte prefix/zero padding, and an integrity digest. Updates and finalization decode a validated copy; updates return a new state and finalization is repeatable. Malformed states raise `INVALID_ARGUMENTS`. File calls use the same engine with fixed 32 KiB binary reads; open/read/close failures raise `NOTREADY`, while filesystem selection, cancellation, and application size ceilings remain caller policy. See the maintained rxhash reference for the exact layout, signal categories, and examples.

`djb2`, `fnv1a`, and `crc32` accept one `.binary`; `murmur3` also accepts an `.int` seed. Each returns the algorithm's unsigned 32-bit bit pattern represented in `.int`. Inputs are not mutated. No historical `rxmath` aliases are retained.

`rxc` records provider ID `rx_hash` in RXBIN metadata for a retained call. `rxvm` and `rxbvm` then find the trusted `rx_hash.rxplugin` automatically, while `crexx -native` selects `rx_hash.a` (or `.lib`) and retains its registration anchor. No Rexx declaration wrapper or explicit runtime provider argument is required in a standard build or installation.

A.4.3 RCC-5D through RCC-5F providers

The remaining RCC-5 providers use the same declarative dynamic/static delivery route:

Provider ID	Public namespace and procedures	Contract note
`rxstats`	`rxstats.mean`, `stddev`, `covariance`, `correlation`, `regression`	Level G statistics over borrowed read-only `.packedfloat` payloads. Compensated shifted-origin accumulation plus a compensated second pass protects ill-conditioned central moments; regression returns immutable `.linearfit`. Boxed arrays, `.packedint`, and raw `.binary` are not production overloads.
`rxvector`	`rxvector.decodef32le`, `encodef32le`, `cosine`, `topkcosine`	Level G exact vector computation over borrowed `.packedfloat`/`.packedint` payloads, with explicit canonical little-endian float32 conversion. The provider is stateless and process-reentrant; prepared/ANN indexes are not part of this contract.
`rxid`	`rxid.uuid4`, `uuid7`, `ulid`, `nanoid`, `snowflake`, `base58`	Bundled optional Level G identifier strings, callable from B when installed; random forms use platform cryptographic randomness and generation failures signal.
`rxfs`	`rxfs.cwd`, `loadpath`, `chdir`, `isdir`, `mkdir`, `rmdir`, `delete`, `rename`, `isfile`, `listdir`, `append`	Narrow filesystem and directory operations. Return/status contracts are documented in the library reference.
`rxplatform`	`rxplatform.uptime`, `user`, `host`, `osname`, `sleep`	Bundled optional Level G host/platform information and millisecond sleep, callable from B when installed. Clipboard, beep, process-global and developer functions from the old draft `system` surface were retired.

Provider ID	Public namespace and procedures	Contract note
`rxunicode`	`version`; four normalization transforms and predicates; `toUppercase`, `toLowercase`; full/simple/Turkic folding; typed `encode`, `decode`, and codec predicates; grapheme count, substring, position, reverse, and `.graphemes`	Level G Unicode 17.0.0 services over `.string` plus explicit `.binary` codec boundaries. The pure Level G facade delegates to private Level B codepoint executors and generated immutable tables. See the Unicode library reference and implementation context.

The source CMake target for `rxplatform` is named `_platform` to avoid a target collision, but `PROVIDER_ID rxplatform` makes its manifest, artifact stem, RXBIN dependency, runtime lookup, and native archive identity consistently `rxplatform`.

A.4.4 `rxsqlite` typed database provider

`rxsqlite` is a standard session-aware RXPA provider rather than an incubator or demo. It owns opaque database/statement native payloads, gives every VM its own handle registry and diagnostic state, and marks its procedures `SESSION_-AFFINE`. Copied payloads retain their resource; explicit close, finalization, stale/wrong-kind/cross-session rejection and session teardown are all provider-owned.

The provider preserves SQLite NULL, signed 64-bit integer, real, UTF-8 text and exact binary blob types. Its public surface covers prepared statements and metadata, modes, busy timeout, WAL checkpoint, bounded online backup and integrity checking, changes/row identity, and primary plus extended errors. The separate `rxsqlite_address.rxbin` Level G module implements SQL commands and named host bindings only through that typed API.

The source-controlled SQLite 3.53.2 amalgamation is compiled with `SQLITE_-THREADSAFE=1`, `SQLITE_ENABLE_FTS5=1`, FULLMUTEX connections and the documented extension/security policy. It is embedded in `rxsqlite.rxplugin` and the canonical `providers/rxsqlite` static archive, so installed native packaging requires no downstream SQLite SDK or extra link metadata. See the maintained rxsqlite reference.

A.4.5 Level G packed numeric owners

The `rxfnsg` REXX library publishes `.packedfloat` and `.packedint` as the comfortable Release 1 Level G owners of the Level B host-native packed numeric surface:

```
import rxfnsg

values = .packedfloat(3)
call values.set(2, 100.0)
say values.get(2)

indexes = .packedint(16)
call indexes.fill(-1)
```

Both classes provide `*(size)`, `fromBinary(data)`, `size()`, `get(index)`, `set(index, value)`, `resize(size)`, `fill(value)`, and `binary()`. Counts and indexes are item-based and zero-based. Construction and growth zero-fill; `fromBinary` is an explicit inbound copy boundary. `binary()` returns a weak mutable `reference .binary`: `dereference` creates a scoped live alias without copying, while `snapshot` is the explicit outbound copy boundary. The packed owner must outlive the reference.

The owned storage is the class object's `register.0.binary` component rather than a separate REXX binary attribute. Native providers may therefore receive a declared `.packedfloat` or `.packedint` argument and consume its payload directly; the caller does not invoke `binary()` and no intermediate REXX byte copy is required. The payload remains host-local and must use the raw encoded binary route for files, persistence, wire formats, or incompatible hosts.

RXPA's `ISINITIALIZED(value)` helper lets such a provider preserve the ordinary typed-object `OBJECT_NOT_INITIALIZED` boundary before borrowing the payload. `ISINITIALIZED` and the later `CALLMETHOD` callback helper are appended to `rxpa_initctx`, so earlier field offsets remain stable, but the initializer context has no negotiated size. These pre-release extensions are therefore a rebuild-together boundary: do not mix a plugin compiled against the current `crexxpa.h` with an older host binary. The compiler scan stub and both dynamic and static VM initializer contexts must populate every appended helper.

The classes deliberately wrap the existing `<packed..float>` and `<packed..int>` Level B instructions. They do not change ordinary `.float[]` or `.int[]` arrays and do not yet provide `x[index]` syntax; both are post-release language work.

A.5 5. Declaring Native Classes and Interfaces

RXPA can also publish class/interface contract metadata to the compiler and VM. Use this when a native or hybrid provider needs to expose the same class-shaped contract that REXX source would normally declare.

```
LOADFUNCS
ADDINTERFACE("demo.environment");
ADDFACTORY("demo.environment", "*", ".environment", "name=.string");
ADDMETHOD("demo.environment", "describe", ".string", "");

ADDCLASS("demo.nativeenvironment");
ADDIMPLEMENTS("demo.nativeenvironment", "demo.environment");
ADDFACTORY("demo.nativeenvironment", "*", ".nativeenvironment", "name=.
    string");
ADDMETHOD("demo.nativeenvironment", "describe", ".string", "");

ADDPROC(make_env, "demo.make", "b", ".environment", "");
ENDLOADFUNCS
```

REXX source factories omit return types (`*: factory`). RXPA declaration macros still carry a return-type string because they emit low-level metadata directly; use the owner contract type for that metadata until RXPA grows an owner-derived factory helper.

Available declaration macros:

- `ADDCLASS(name)` and `ADDCLASSX(name, option, type)`
- `ADDINTERFACE(name)` and `ADDINTERFACEX(name, option, type)`
- `ADDIMPLEMENTS(class_name, interface_name)`
- `ADDFACTORY(owner, member, return_type, args)`
- `ADDMETHOD(owner, member, return_type, args)`
- `ADDDEFAULTMETHOD(owner, member, return_type, args)`
- `ADDMEMBER(owner, kind, member, return_type, args)` for the generic form

These declarations are consumed from both dynamic `.rxplugin` modules and static `DECL_ONLY` declaration libraries. They make contracts visible for type checking and runtime metadata discovery.

A.5.1 Calling back into a CREXX object

A native procedure can synchronously invoke a method on an object received from CREXX with `CALLMETHOD`:

```
PROCEDURE(invoke)
{
    rxpa_attribute_value method_args[1];
    method_args[0] = ARG1;
    if (CALLMETHOD(ARG0,
                   "rxsig1|on_value|.int|value=.int",
                   1, method_args, RETURN) != 0) {
        return; /* CALLMETHOD has populated SIGNAL */
    }
    RESETSIGNAL
}
```

The receiver must be a live, initialized CREXX object. The descriptor is the canonical `rxsig1|name|return_type|arguments` form and is checked against the concrete receiver class at runtime. The argument array and result are borrowed RXPA handles; the call copies the result into the supplied result slot and copies receiver/argument mutations back before it returns. `CALLMETHODX` is the same operation with an explicit signal handle for native callbacks outside the lexical `PROCEDURE` body.

An unhandled signal raised by the CREXX method is propagated through the RXPA signal handle. Ordinary non-zero `.int` and `.boolean` method results remain ordinary results rather than being confused with signal status codes.

The call is same-thread, synchronous, and valid only while an RXPA procedure is active. A plugin must not retain the receiver, argument, result, or signal handles after that outer call returns. The VM preserves the outer external-call trampoline and recursively enters the existing worker lifecycle, so the CREXX method may itself call RXPA procedures. Legacy plugin calls use the recursive compatibility lane, which prevents that nested native re-entry from deadlocking.

Declaration is not construction. `ADDCLASS`, `ADDINTERFACE`, `ADDIMPLEMENTS`, and the member macros tell the compiler and VM that a contract exists, but they do not by themselves run a factory or stamp class identity on a return value. Existing RXPA return helpers such as `SETSTRING`, `SETINT`, and array attribute helpers fill a return value slot; factory/class construction is still a separate operation.

That means a native procedure can advertise a typed signature, for example:

```
ADDPROC(make_env, "demo.make", "b", ".environment", "");
```

but the C body must still create or receive an object value that has the right shape and class identity. For complete object creation today, use a small Rexx factory/class shim to create the typed Rexx object, and let that object delegate selected work to native C functions. A future RXPA helper should cover the pure-C operation of constructing/stamping a typed object directly.

Ordering matters when a native procedure signature references a class or interface type. Put the relevant `ADDCLASS`/`ADDINTERFACE` metadata before the dependent `ADDPROC` so the compiler knows the type before it validates the procedure signature:

```
LOADFUNCS
ADDINTERFACE("demo.environment");
ADDMETHOD("demo.environment", "describe", ".string", "");

ADDPROC(make_env, "demo.make", "b", ".environment", "");
ENDLOADFUNCS
```

A.5.2 Native payload ownership

When a native implementation needs to hide a C-side handle inside a Rexx object, the preferred physical storage is the value's binary payload. Attach shared static payload operations with `SETNATIVEPAYLOAD()` if the payload owns native resources:

```
static const rxpa_native_payload_ops env_payload_ops;

static void env_finalize(rxpa_attribute_value value) {
    EnvHandle **slot = (EnvHandle **)GETNATIVEPAYLOAD(value, NULL, NULL
        , NULL);
    if (slot && *slot) env_release(*slot);
}

static void env_copy(rxpa_attribute_value dest, rxpa_attribute_value
    source) {
    EnvHandle **slot = (EnvHandle **)GETNATIVEPAYLOAD(source, NULL,
        NULL, NULL);
    EnvHandle *copy = slot && *slot ? env_retain(*slot) : NULL;
    SETNATIVEPAYLOAD(dest, &copy, sizeof(copy), &env_payload_ops, 0);
}

static const rxpa_native_payload_ops env_payload_ops = {
    "demo.EnvHandle",
```

```
    env_copy,
    env_finalize
};
```

The ops object is provided by the native module and shared across instances; the value stores only a pointer to it. The VM owns the per-value binary payload buffer. `SETNATIVEPAYLOAD()` mallocs VM-owned storage, copies the supplied payload bytes into it, and records the shared ops pointer and flags. Copy hooks must install the destination payload through `SETNATIVEPAYLOAD()` rather than storing externally allocated memory directly in the destination value. On `clear_value()`, the finalizer runs before the VM frees the binary buffer; the finalizer releases nested native resources but must not free the payload buffer itself. On `move_value()`, the buffer and ops pointer move together. On `copy_value()`, the VM calls the copy hook if set; if it is not set, the VM byte-copies the payload and copies the ops pointer. That fallback is only safe when the payload was deliberately designed to tolerate duplicate finalization, such as a registry handle or refcounted pointer. Use `RXVM_NATIVE_PAYLOAD_FLAG_BITCOPY_SAFE` to document that intent on such payloads.

A.6 cREXX Level B `stem` class

This document defines the stable Level B `.stem` and `.stemIterator` contract.

A.7 Overview

A `stem` is a string-to-string dictionary supporting CREXX property and bracket notation for Classic-style compound-variable tails.

A missing key returns the assigned stem default, or the empty string before a default is assigned.

Use `size()` or the key arrays when the absence of a key must be distinguished from an explicitly assigned empty value.

A.8 API

- `get(key = .string) = .string` returns the tail value, the stem default, or the empty string.
- `set(key = .string, value = .string) = .void` inserts or replaces a value. Property assignment with an omitted key, `stem. = value`, assigns the Classic stem default instead; an explicitly supplied empty key is still treated as a normal key. Assigning a default also replaces all values belonging to existing tails.
- `size() = .int` returns the number of distinct keys.
- `key(index = .int) = .string`, `value(index = .int) = .string`, and `valueAt(index = .int) = .string` access the internal one-based insertion arrays. An index outside `1..size()` signals `INVALID_ARGUMENTS`.
- `tails() = .string[]` and `values() = .string[]` return independent arrays.
- `iterator() = .stemIterator` creates a live iterator.
- `snapshotIterator() = .stemIterator` creates a snapshot.

`get()` performs lookup by tail name, whereas `key()`, `value()`, and `valueAt()` operate on insertion-order indexes.

The iterator factory accepts only snapshot flags `0` and `1`; any other flag signals `INVALID_ARGUMENTS`.

A.9 Implementation (hash map with separate chaining)

The implementation uses parallel arrays to manage a 256-bucket hash map:

- `buckets`: an array of 256 integers pointing to the head of a chain.
- `keys`: a flat array storing string keys.
- `vals`: a flat array storing string values.
- `value_generations`: the default generation in which each explicit value was assigned.
- `next`: an integer array storing the index of the next item in the chain.

Stem-default assignment is constant time. It stores the new default and advances the default generation; older explicit values then read as the default without rewriting the value array.

A later explicit tail assignment records the current generation and becomes visible again.

Indexed access, extracted values, and iterators all resolve the effective value through the same generation rule.

The hash loop reads Unicode codepoints directly with VM string instructions and uses a multiplier of 31.

Each step is reduced modulo 256, which is equivalent for the final bucket and prevents native integer overflow for long keys.

A.10 Usage

Stems can be accessed using standard `get()` and `set()` methods, but the preferred approach is to use CREXX object property syntax (`obj.key` or `obj["key"]`), which is automatically rewritten to method calls by the compiler.

```
s = .stem()

s. = "fallback"
say s.unknown              /* fallback */

/* Property syntax (syntactic sugar) */
s.key = "value"
val = s.key

/* Multi-tail property syntax preserves separators in the key */
customer = "acme"
invoice = "2026.05"

s.customer.invoice = "value4"   /* key is "acme.2026.05" */

/* Bracket notation for keys that aren't valid identifiers */
s["my-key"] = "value2"

/* Direct method calls */
call s.set("other_key", "value3")
val = s.get("other_key")
```

A.11 Iteration

Stems are keyed containers, so iteration is over tails/keys rather than list positions.

Iteration order is unspecified.

The current implementation stores keys in insertion order, but callers must not depend on that ordering.

`iterator()` returns an unsynchronized live iterator.

It observes the current stem through a weak reference; value updates after iterator creation are visible, and newly added tails may be observed by the current implementation.

`snapshotIterator()` copies the stem's tails and values once in the iterator factory.

Use it when callers need stable factory-time contents while the stem may be mutated.

The iterator API is:

- `hasNext() = .int`
- `next() = .string`, returning the next tail/key
- `value() = .string`, returning the value for the current tail
- `index() = .int`
- `reset() = .void`
- `isLive() = .int`

A.11.1 Iterate with an iterator

```
s = .stem()

s.name = "Ada"
s.lang = "Rexx"

it = s.iterator()

loop while it.hasNext()
    tail = it.next()
    say tail "=" it.value()
end
```

A.11.2 Iterate with `tails()`

`tails()` returns an independent `.string[]` containing all tails.

Use `get()` to retrieve the value belonging to a tail.

```
countries = .stem()

countries.Germany = "Berlin"
countries.France  = "Paris"
countries.Italy   = "Rome"

tails = countries.tails()

do j = 1 to tails[0]
    k = tails[j]
    say k "=" countries.get(k)
end
```

A.11.3 Iterate with `tails()` and `values()`

If both keys and values are required, extract both arrays once.

```
tails  = countries.tails()
values = countries.values()

do j = 1 to tails[0]
    say tails[j] "=" values[j]
end
```

A.11.4 Indexed access

`key()`, `value()`, and `valueAt()` operate on the internal one-based insertion arrays.

They expect a numeric index, not a tail string.

```
say countries.key(1)
say countries.value(1)
say countries.valueAt(1)
```

The following is incorrect because `value()` expects an insertion index rather than a tail.

```
k = "Germany"
say countries.value(k)      /* WRONG */
```

Retrieve values by tail using `get()`.

```
k = "Germany"
say countries.get(k)        /* CORRECT */
```

A.12 Performance

Hashing uses direct `strlen` and `strchar` VM operations and makes no Level B selector calls.

Lookup and update use separate bucket chains with early exit.

The implementation remains Level B so it can evolve with the standard typed array surface; no native-C replacement is required by this contract.

A.13 Notes on Character Encoding

VM string parsing exposes Unicode codepoints rather than UTF-8 bytes, so equal Unicode keys hash consistently regardless of their encoded byte width.

Bibliography

Cowlishaw, M. F. (1985). *The REXX language: a practical approach to programming*. Prentice-Hall, Inc.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1993). "Design patterns: Abstraction and reuse of object-oriented design". In: *ECOOP'93—Object-Oriented Programming: 7th European Conference Kaiserslautern, Germany, July 26–30, 1993 Proceedings 7*. Springer, pp. 406–431.

Knuth, D. E. (1998). *The Art of Computer Programming: Sorting and Searching, volume 3*. Addison-Wesley Professional.

List of Tables

Index

ISBN 978-94-039116-4-9

9 789403 911649 >