The `nigeria` Character Set Conversion and Text Processing Library

Guide to Installation, Configuration, and Use (library version A.0.0.4)

Bionic Buffalo Corporation

The `nigeria` Character Set Conversion and Text Processing Library: Guide to Installation, Configuration, and Use (library version A.0.0.4)

by Bionic Buffalo Corporation

Table of Contents

List of Tables

Abstract

This is the documentation for the **nigeria** library, which includes functions for character set conversion and text processing. **nigeria** provides an API to enable programs to handle internal text processing in Unicode, using common string operations such as sorting, searching, translation, and string duplication.

nigeria also provides functions to convert to and from other popular, non-Unicode character sets and encoding schemes.

Chapter 1. Introduction

1.1. Background

For centuries, typemakers and printers have designed collections of graphics which were used to represent various letters and symbols used in books, newspapers, and other printed material. In each context, certain graphics were almost always expected: the letters of the alphabet, digits, and common punctuation symbols such as periods and commas.

In some cases, educators formally or informally organized graphics into collections to be used as the basis for the instruction of students or use by citizens. For example, the Japanese government promulgated the *Toyo Kanji*, a standard set of ideographic characters for everyday use, especially as a standard for schools.

When computers came along, they were originally used primarily for computation. Eventually, though, it became desirable to store and to manipulate textual information in computers. To do this, mappings from the graphics used by humans to the codes used inside computers was necessary. To make the mappings useful, it was also necessary that they include codes to represent non-graphics such as printer line feeds and end-of-text markers. This mapping requirement was complicated later when data communication developed, giving rise to the need for additional symbols to represent communications functions such as message acknowledgements and error indications.

These mappings took several forms. First, of course, were simple correspondences between numbers in storage and the graphic symbols or control functions. (For example, the early *BCDIC* code assigned **`0x31`** to the letter *A*, **`0x32`** to the letter *B*, and so on.) Second, there were special mappings for physical media. (IBM's *EBCDIC* code for punched paper cards designated the letter *A* by a punch in the 12-row along with a punch in the 1-row.)

At first, storage and bandwidth were at at steep premiums, so it was important to condense the codes as much as possible. Whereas a typical lead type set from a type foundry might include separately-designed italic characters, the early encoding schemes were restricted to the basic alphabet. A legacy of this development is that type foundries thought of the letter *A* and the letter *italic A* as two separate letters,

where most computer encoding schemes consider an *italic A* to be simply a variant of the basic letter *A* . Some of the very earliest mappings didn't distinguish between upper and lower case letters, either.

Different vendors developed different mappings, and countries with various languages and alphabets also created variants. Eventually, dozens of special mappings developed, and are still in frequent use.

The numerous graphic and control codes to be mapped created a serious problem for some mappings. Most software was restricted to handling text consisting only of one-byte numbers, so the number of codes in a mapping was limited. To circumvent the restriction, large mappings used sequences of codes for a single character, or used one or more special symbols to select from among multiple mappings encoded into the same stream. Common examples of this approach are found in the *EUC*, *Shift-JIS* , and *Big-5*. This technique is unavoidable especially in many Asian character sets with thousands of ideographs.

Meanwhile, there were some other mappings developed based on 16-bit and larger codes for internal use by software, with transformations to and from single-byte encoding forms for communication and storage. These included a few schemes to create a single, unified mapping for all languages.

Although the idea of a unified mapping scheme was attractive, it faced numerous obstacles. These include technical problems (how to merge right-to-left Arabic into left-to-right German), economic issues (the cost of implementing a new scheme while interoperating with old systems), business disputes (many firms didn't want to change from proprietary character sets), linguistic differences (Japanese and Chinese dictionaries collate the same ideographs in different sequences) and political challenges (such as the disunity between China and Taiwan).

Eventually, however, the difficulties were overcome, and a new scheme called Unicode was developed. Unicode was incorporated into an ISO standard (ISO 10646), and has been adopted by most major operating systems developers. Although most application software still does not implement Unicode, it is clearly the first choice for developers wanting to support multiple languages in their applications. (Note: *Unicode* is a registered trademark of Unicode, Inc.)

1.2. Basics of Unicode

Unicode represents each character by a single number. It maps more than 90,000 characters to unique numbers, so even 16-bit numbers are not adequate to represent the code values. On the other hand, it specifies transformations between these numbers and 7, 8, or 16-bit code values, so byte streams can be used to represent Unicode characters.

Not only letters, digits, punctuation, and ideographs are mapped, but also graphical, line-drawing, musical, mathematical, scientific, and other symbols. For the most part, the designers of Unicode tried to accomodate all of the symbols used in other character sets. For instance, since the common PC character set used by IBM included certain line-drawing characters, those line-drawing characters may be found in Unicode.

Even though a character may be used in more than one language, it is defined only once in Unicode. For example, the letter **`LATIN CAPITAL LETTER A`** is mapped once, even though it is used in English, German, and (as *romaji*) in Japanese. On the other hand, it was decided that **`CYRILLIC CAPITAL LETTER A`** is a different character from **`LATIN CAPITAL LETTER A`**, even though the two letters look a lot like each other. The reasoning behind decisions like these is interesting to linguists, but usually not important to programmers.

To facilitate use of Unicode, and interoperation with other character sets, the Unicode Consortium also publishes conversion tables to convert among Unicode and these other mappings. *Every major character set can be mapped to Unicode, but the converse is not true.* When converting from Unicode to some other character set, there are Unicode characters which cannot be mapped. A programmer must make provisions for dealing with these inconvertible code values.

Unicode reserves certain code values for proprietary characters, so an implementation may define new character mappings. These may be used for alphabets or ideographs not yet defined, or they may be used by firms for corporate symbols. Use of proprietary characters carries the risk that interoperability with other implementations may be reduced.

The mappings defined by Unicode are not static. Occasionally since the first version of Unicode was released, the Unicode Consortium has released updated versions of the standard, which map more and more characters.

Each Unicode character is defined to have certain properties. Not all properties are

defined every character, however. For examples, the ideographs do not have names, and only a minority of characters have numeric values. The main properties of the characters are:

- name: a unique name assigned to the character
- numeric values: the character's numeric values (there are three kinds of numeric value: decimal digit, other digit, and other numeric value)
- category: letter, digit, symbol, etc.
- directionality ("directional category"): shows whether the character has intrinsic left-to-right or right-to-left attributes
- case: mapping to uppercase, lowercase, and titlecase equivalents
- decomposition: the mapping to simpler components (as **`LATIN CAPITAL LETTER A WITH CIRCUMFLEX`** decomposes into **`LATIN CAPITAL LETTER A`** followed by **`COMBINING CIRCUMFLEX ACCENT`**)
- combining class: designates how certain characters interact typographically

There are also other properties assigned to the Unicode characters, but these are the most important for most processing.

The Unicode standard does *not* define any specific collating or sorting sequence, although it provides guidance to developers wanting to implement collating sequences.

1.3. Encoding Schemes

Unicode and other character sets use various encoding schemes to represent their code values. It is ambiguous to specify one without the other, although sometimes the common usage makes one interpretation probable. For instance, Shift-JIS encoding is used almost exclusively with with the JIS character set, so a specification of "Shift-JIS" is usually specific enough. Similarly, in the context of the web, practically all Big-5 pages are encoded the same way, even though other ways are possible, so browsers offer only, simply "Big-5" as a choice.

What follows is a list of some common encoding schemes.

- *simple byte encoding:* This method is used for character sets (such as ASCII or Latin-1) which have 256 or fewer codes. Each code is represented as a single byte.
- *UTF-8:* This method encodes values less than 128 as single bytes, and values at or above 128 as sequences of bytes. Usually used with Unicode, and common on the web.
- *Shift-JIS:* This method encodes values less than 128 as single bytes, and values at or above 128 as pairs of bytes. Usually used with JIS X 0208, a common Japanese character set. (Developed by a consortium led by Microsoftware Associates, a former Japanese software company sometimes confused with Microsoft, but first widely implemented by Microsoft.)
- *EUC:* Also called Extended Unix Code, or ISO 2022, EUC is used for a variety of character sets including Chinese and Japanese mappings, and comes in two forms. One form encodes all characters as pairs of bytes (16 bits together), and the other form encodes characters as sequences of from one to four bytes. An important aspect of EUC is that it can dynamically select character sets, so a sequence might mean one character at one time, and a different character at another. Because multiple character sets can be encoded together, it is necessary to know something about the previous codes in a stream to understand the current codes.
- *UCS-2:* This method uses 16-bit codes, and is almost always associated with Unicode characters. Characters are encoded as one or two code values.

Externally, the **`nigeria`** supports these and other encoding schemes. Internally, the library encodes a character as type **`civ_superchar_T`**, which is defined as a **`signed long`**. Positive values represent Unicode characters; negative values are used for internal purposes.

Chapter 2. Concepts and Conventions

2.1. Traditional String Operations

C programmers commonly encode text as *strings* , **`NULL`** or zero terminated arrays of **`char`**. Newer C compilers offer a type **`wchar_t`** for "wide" characters, but this type isn't very portable. (This is partly because some implementations use **`char`** to implement **`wchar_t`**, obviously not very useful for large character sets. Incredibly, the C standard allows this.)

Standard C libraries provide a number of procedures to operate on **`char`** strings. These common routines include **`strlen()`**, **`strcat()`**, **`strcmp()`**, and others. Equivalent procedures for wide-character strings are **`wstrlen`**, **`wstrcat()`**, **`wstrcmp()`**, and so on.

Many libraries also support "multibyte" strings, which are made of **`char`**, but are interpreted according to an encoding such as EUC or Shift-JIS so that a single character may be represented as a sequence of more than one byte. Procedures to support multibyte strings typically have names such as **`mbstrlen()`** or **`mbstrchr`**. Note that the implementation of these procedures is highly dependent on the encoding scheme used for the multibyte strings.

Typically, a number of conversion routines are also available. For instance, **`wctomb()`** converts a wide character to a multibyte character.

Most of these procedures have no knowledge of the character set used in the string. The **`strchr()`** procedure (which searches for a specific code in a string) works the same if the string and code are ASCII or EBCDIC or Latin-2 characters. Other procedures (such as **`isalpha()`**) presume a specific, if not selectable, character set is used for encoding.

Some programming languages (such as Java and Perl) make stronger assumptions about the character sets and encoding schemes used in strings. The **`nigeria`** library similarly makes strong assumptions for C programs: the character set is always Unicode, and encoding is always as type **`civ_superchar_T`** , or **`long`**. (The type **`civ_superchar_T`** is defined to allow for stronger type checking by programs similar to **`lint`**, to remind the programmer about the semantics and syntax of variables, and to allow redefinition on machines where a **`long`** is 64 bits

and a shorter type would be more efficient.)

The **nigeria** routines used for string and other text processing, along with traditional **char** equivalents, are summarized below.

Table 2-1. nigeria Text Manipulation Routines

nigeria Procedure	**Description and Equivalent Traditional Procedure**
nga_string_break()	search a string for any of a set of characters (**strpbrk()**)
nga_string_compare() **nga_string_compare_counted()**	compare two strings using simple numeric collating sequence (**strcmp() strncmp()**)
nga_string_complement_span()	find the length of a string consisting of characters not in a second string (**strcspn()**)
nga_string_concatenate() **nga_string_concatenate_counted**	concatenate two strings (**strcat() strncat()**)
nga_string_copy() **nga_string_copy_counted()**	copy a string into a buffer (**strcpy() strncpy()**)
nga_string_duplicate() **nga_string_duplicate_counted()**	make a copy of a string (**strdup() strndup()**)
nga_string_length()	determine length of string (**strlen()**)
nga_string_scan_char()	scan a string for a character (**strchr()**)
nga_string_scan_char_reverse()	scan a string in reverse for a character (**strrchr()**)
nga_string_span()	find the length of a string consisting of characters in a second string (**strspn()**)
nga_string_substring()	locate a substring within a string (**strstr()**)

Detailed descriptions of these functions can be found in the API reference.

2.2. Translation and Transliteration

Translation among character sets is complicated by several factors. Most of these difficulties occur when translating from Unicode to some other character set, because Unicode is a proper superset of the other sets.

The source character may not have an equivalent in the target set. The **`nigeria`** translation routines will insert a substitute string into the destination, in lieu of the translated source character. Substitute strings may be fixed values, or may be derived from the source character name, code value, pronunciation, or some near typographical equivalent of the source character.

In some cases, a single code is used where multiple codes might be more semantically precise. For example, in ASCII, people often write “(c)” because there is no copyright symbol “©”. When translating from ASCII to Latin-1, how is a translator to know if the string “(c)” should be translated as a copyright symbol or as a sequence of three separate characters? This library will translate each character separately to its nearest equivalent, without regard to context.

Although characters may translate one-to-one, this is not always true of codes. A character may be represented by a sequence of codes, so the length of the source string will not always be a simple multiple or fraction of the length of the translated string. Moreover, in some character sets and encoding schemes, the meaning of a code may depend on the previous codes in the string, so it is not possible simply to break an input string into separate characters and to translate each character in isolation.

Sometimes translation must operate with incomplete data. For example, when translating a file, the contents are read into memory a buffer at a time. The read routines do not know if they have read in whole characters, or merely some of the codes of a multi-code character. Each translation operation has two procedures: one to initialize the translation operation, and another to continue the operation. The initialization routine creates a state variable which is passed to the continuation routine. The continuation routine is called repeatedly, each time with more data. When there are no more data, a flag is passed to the continuation routine by the caller, and the translation will be completed.

The **`nigeria`** library implements translation to and from Unicode. It will not translate directly among other character sets. In order to translate from one non-Unicode set to another non-Unicode set, it is necessary first to translate to

Unicode.

2.3. Collation and Comparison

There are many collating sequences, even within a given language. In English, for example, the collating sequence for telephone directories is not always the same as the sequences for libraries or dictionaries. Therefore, collation is dependent on context.

Ordering a collection of text strings is based on the ability to compare any two given strings, and to decide which one comes first and which one second. Sorting is based on repeated application of this two-string decision process. However, string comparison in not always based on the simple comparison of two characters. For example, in French, later characters in a string sometimes have more importance than earlier characters.

Simple alphabetical order is not always applicable. Some languages order the same alphabet differently than others, other languages use different alphabets, and some languages such as Chinese don’t use an alphabet at all. Often, when foreign phrases are sorted into a language, they are collated among local phrases based on transliteration. This can result, for instance, in English text collating in Russian alphabetic order.

Ideographic characters are collated differently in different circumstances. Chinese dictionaries come in several different orders, and Japanese dictionaries sometimes collate Chinese characters according to their Japanese pronuciation (using Japanese alphabetical order). (Ideographic dictionaries often have indices, so a reader can look up a character by pronunciation, radical, structure, or stroke count.) European-language libraries often collate Chinese characters into local languages by pronunciation, but each Chinese character has multiple pronunciations (by dialect) and each pronunciation has several spellings (by romanization method).

This library will not handle all collation schemes. However, it will handle many important ones, and will be expanded in the futures.

Standard C defines *locales*, which encapsulate a view of collating sequences, dates, currency representations, and other information. Unfortunately, the locale is not usually adequate to represent the needs of the user, especially when multiple

languages are involved. For example, the locale for France does not tell how to collate a Chinese phrase in a French document.

`nigeria` uses options to specify a collation sequence, instead of using a simple locale. Options are implemented as ascii text strings, and may be combined. When options conflict in a string, the later option takes precedence, so an existing option string may be modified merely by concatenating a string containing an overriding options. Some common locales are provided as predefined string constants, so a user may start with a standard locale and then modify the behaviour as desired.

2.4. Property Queries

The Unicode character properties can be queried for any code value by use of a series of procedures.

`nga_categorize_character()` returns the general category of a character.

`nga_character_name ()` returns the official Unicode name of a character.

`nga_decompose_character()` returns the decomposition of a character into components.

2.5. Miscellaneous Considerations

2.5.1. Fonts

Although fonts are sometimes confused with character sets, they are distinct entities. A font is a rendering of the graphical characters in a character set, so each font is associated with some character set.

Sometimes it is possible to “fool” a system by using a foreign font with a local character set. For example, a document whose character set is Latin-1 (ISO 8859-1) can include some Russian words (Cyrillic characters) by using a Cyrillic font for those words. However, the underlying software still thinks the document is in Latin-1.

It is fine to do this kind of thing in a pinch when the correct character set support is not available. However, this can lead to serious problems in some circumstances.

If you are having problems using various fonts and character sets, it may be because the font's character set is not the same as the document's character set.

2.5.2. Dynamic Memory

Although most software dynamically manages memory using **`malloc()`** and **`free()`**, there are situations where this is not desirable. Especially, some systems use different dynamic memory schemes for different types of data. For example, some operating systems distinguish between kernel memory and user memory, and some run-time environments allocate string memory from a special region to permit performance optimization.

All dynamic memory management in the **`nigeria`** library uses **`nga_memory_allocate()`** and **`nga_memory_free()`**. As delivered, these routines are implemented using the standard **`malloc()`** and **`free()`** procedures. However, the user may replace those routines as desired to support specialized memory management systems.

Chapter 3. Coding Examples

TBS

Chapter 4. Installation and Configuration

4.1. Obtaining the `nigeria` Distribution

The **`nigeria`** library is written and maintained by Bionic Buffalo Corporation. It is distributed on the internet. On the web, the home page is **`http://www.tatanka.com/prod/info/nigeria.htm`**. On the home page are pointers to HTTP and FTP servers which carry the distribution.

The library is distributed in **`.tgz`** and in **`.zip`** formats. Use the most convenient form for your development environment.

The **`ivory coast`** software, which consists of header files and a small library, are also needed to build **`nigeria`**. You need the header files only for **`nigeria`**; the library is not required. There is a pointer to **`ivory coast`** on the **`nigeria`** home page.

4.2. Distribution Contents

The distribution archive contains documentation and source code only. There are no compiled object files or libraries. The source code is standard C.

See the **`README`** file in the top directory of the archive for details.

The Bionic Buffalo software, including **`nigeria`**, **`ivory coast`**, and other programs is most easily built by creating a common directory for the various Bionic Buffalo distributions. Each archive is then unpacked into its own subdirectory. The top directory can be named anything convenient. If the top directory is called **`buffalo`**, then you will have directories **`buffalo/ivoire/`**, **`buffalo/nigeria/`**, and so on. The **`make`** operation may create additional, parallel directories (such as **`buffalo/inc/`** and **`buffalo/lib/`**). By keeping the Bionic Buffalo software in a separate directory tree such as this, the **`make`** is unlikely to inadvertently overwrite existing files.

4.3. Configuration

There is no special configuration process required for **`nigeria`**, which acquires its configuration from **`ivory coast`**. Configure and install **`ivory coast`** first; refer to its documentation for details.

4.4. Building the Library

The distribution contains Makefiles for the gnu tools only. If you are building under another environment, you will need to create your own makefiles or project files.

There are no special operations needed to build the library. The process consists only of simple compilation and putting the object files into a library file.

(Note for users of other **`make`** programs: Instead of using **`make depend`** to determine the dependencies, we use dependency files with the suffix **`.d`**, which are then **`INCLUDE`**d into the **`Makefile`**. That is the purpose of the rule at the top of the **`Makefile`**, which creates **`.d`** files from **`.c`** files.)

The distribution includes a few standalone test programs which are to be linked with the library. These include **`ngaquery`**, which will print the Unicode properties of a character. You can use these programs to verify that the libaries built correctly.

Chapter 5. API Reference

5.1. `nigeria` API

5.1.1. `nga_categorize_character()`

Determines the Unicode general category of a character.

Synopsis.

```
civ_status_T    nga_categorize_character
                        ( civ_superchar_T       code,
                        short                 * category ) ;
```

Arguments.

`code`

(in) a Unicode character value.

`category`

(out) the general category of the given Unicode character. Possible values are defined as constants **`NGA_CATEGORY_*`**.

Returns. Standard exceptions only.

Description. Given a character **`code`**, returns in **`category`** the general category of the character, as defined by the Unicode specification. The possible values of **`category`** allow for undefined characters, so there is no separate exception for invalid input.

5.1.2. `nga_character_name ()`

Determines the official Unicode name of a character.

Synopsis.

```
civ_status_T nga_character_name
                        ( civ_superchar_T       code,
                        unsigned char        ** name,
                        unsigned              * length ) ;
```

Arguments.

`code`

(in) a Unicode character value.

`name`

(in/out) a buffer to contain the name of the character. If **`* name == NULL`**, then a new buffer is allocated; otherwise, the caller-provided buffer is used. Any buffer provided by the caller is assumed to be large enough to hold the output. The name is followed by a terminating **`NULL`** character.

`length`

(out) the length of the character name, not counting the terminating **`NULL`** character.

Returns. Standard exceptions only.

Description. Given a character **`code`**, returns in **`name`** the official Unicode name of the character, as defined by the Unicode specification. The length of the name is returned in **`length`**. If there is no name for the character, or if the character is undefined, then a length of zero is returned.

Notes.

1. Unicode does not assign names to control characters. However, this routine returns names for the control characters. If a control character has a name defined by ISO 6429, then that name is used. Otherwise, the name from RFC 1345 is used.
2. There is no distinction between undefined characters, ideographic characters, and certain other values which do not have names. This routine should not be used to determine if a code is valid. Use **`nga_categorize_character()`** for that purpose.

3. The routine **`nga_longest_character_name()`** can be called to determine the longest possible name, so any caller-allocated buffer will be large enough to hold the result.

5.1.3. `nga_collate_setup ()`

Prepare for one or more collation decisions.

Synopsis.

```
civ_status_T    nga_collate_setup
                        ( unsigned char       * options,
                        nga_status_T          * status ) ;
```

Arguments.

`options`

(in) collation options.

`status`

(out) a status buffer which is retained for multiple invocations of **`nga_collate_decide()`**.

Returns. Standard exceptions only.

Description. Prepares for one or more collation decisions. The caller provides various collation options in a character string, and this procedure initializes the **`status`** for subsequent invocation of **`nga_collate_decide()`**. The process of interpretation of the option strings is relatively expensive, so it is separated from the collation decision operations themselves. This allows the option strings to be interpreted once, then multiple decisions can be made based on that one interpretation.

Notes.

1. If there are two or more options which conflict or contradict each other, then the last option in the string takes precedence.

2. This procedure is not implemented in this version of the **`nigeria`** library. A future version will implement this procedure.

5.1.4. `nga_collate_decide ()`

Decide which of two strings collates before the other.

Synopsis.

```
civ_status_T    nga_collate_decide
                        ( nga_status_T        * status,
                        civ_superchar_T       * string1,
                        civ_superchar_T       * string2 ) ;
```

Arguments.

`status`

(in/out) status structure initialized by a previous call to **`nga_collate_setup()`**. The result of the decision is returned in this structure by setting one and only one of the bits **`status -> greater`**, **`status -> equal`**, or **`status -> lesser`**.

`string1`

(in) the first string to be collated.

`string2`

(in) the second string to be collated.

Returns. Standard exceptions only.

Description. Decides which of two strings collates first. In a previous call to **`nga_collate_setup()`**, the caller has specified options to indicate the algorithms used for the collation.

Notes.

1. When **`nga_collate_setup()`** is called, data structures will be allocated to manage the collation decision process. Before calling **`nga_collate_decide()`** for the last time, the caller should set **`status -> free_state`** to cause those data structures to be released.
2. This procedure is not implemented in this version of the **`nigeria`** library. A future version will implement this procedure.

5.1.5. `nga_decompose_character ()`

Determine the decomposition of a Unicode character.

Synopsis.

```
civ_status_T    nga_decompose_character
                        ( civ_superchar_T       code,
                        civ_superchar_T      ** components,
                        unsigned              * count ) ;
```

Arguments.

`code`

(in) the character to be decomposed.

`components`

(out) the sequence of characters or special values constituting the decomposition. This sequence is *not* terminated by a **`NULL`** character. The special values are defined by the constants **`NGA_DECOMP_FMT_*`**. If **`* components == NULL`**, then a new buffer will be allocated. Otherwise, the caller's buffer is used, and is assumed to be large enough for any possible decomposition.

`count`

(out) the number of components in the decomposition. A value of zero is returned if there is no decomposition defined for the **`code`** value.

Returns. Standard exceptions only.

Description. Determines if a character has a decomposition defined by the Unicode standard, and what that decomposition is.

Notes.

1. A **`count`** of zero is returned for undefined or invalid characters, as well as for characters with no defined decomposition. This routine should not be used to determine if a code is valid. Use **`nga_categorize_character()`** for that purpose.
2. The routine **`nga_longest_decomposition()`** can be called to determine the longest possible name, so any caller-allocated buffer will be large enough to hold the result.

5.1.6. `nga_longest_character_name ()`

Determine the longest possible character name which can be returned by **`nga_character_name ()`**.

Synopsis.

```
civ_status_T    nga_longest_character_name
                        ( unsigned            * length ) ;
```

Arguments.

`length`

(out) the longest character name which can be returned by **`nga_character_name ()`**.

Returns. Standard exceptions only.

Description. Determines the longest possible character name. The returned value does not include the trailing **`NULL`** character.

5.1.7. `nga_longest_decomposition ()`

Determine the longest possible decomposition which can be returned by **`nga_decompose_character ()`**.

Synopsis.

```
civ_status_T    nga_longest_decomposition
                        ( unsigned            * length ) ;
```

Arguments.

`length`

(out) the longest decomposition which can be returned by **`nga_decompose_character ()`**.

Returns. Standard exceptions only.

Description. Determines the longest possible decomposition. The returned value is a count of characters.

5.1.8. `nga_mandarin_pronunciation ()`

Determine the Mandarin pronunciation of an ideograph.

Synopsis.

```
civ_status_T    nga_mandarin_pronunciation
                        ( civ_superchar_T       code,
                        unsigned                method,
                        unsigned              * count,
                        nga_pronunciation_T  ** pronunciations ) ;
```

Arguments.

`code`

(in) the code value of the ideograph to be pronounced

`method`

(in) the method used to represent the pronunciation, selected from one of the constants **`NGA_MANDARIN_*`**.

`count`

(out) the number of pronuciations returned

`pronunciations`

(out) the Mandarin pronunciations of the ideograph

Returns. Standard exceptions only.

Description. Determines the Mandarin romanization of an ideograph. If the romanization is not known, or if the character is not a Chinese character, then **`count`** is set to zero.

Notes.

1. Mandarin is one of the dialects of Chinese. A given ideograph may have a different pronunciation in different dialects.
2. Tone marks are not included in the pronunciations. No composite characters are decomposed.
3. The possible characters used to represent the pronunciation depend on the **`method`** selected. Where latin characters are used, the lower case letters are employed.

   Pinyin romanization is based on the lower case latin letters, plus **`LATIN SMALL LETTER U WITH DIAERESIS`**.

   Wade-Giles romanization is based on the lower case latin letters, plus **`LATIN SMALL LETTER U WITH DIAERESIS`**, **`LATIN SMALL LETTER E WITH CIRCUMFLEX`**, and **`APOSTROPHE`**.

   Yale and *gwoyeu romatzyh* romanizations are based on the lower case latin letters only.

   Bopomofo pronunciations are written using special characters. These are found in the Unicode block beginning at **`0x3100`**.

4. Where multiple pronunciations are possible, no inference is to be made regarding the sequence of values returned. They are in no particular order. In future releases, the sequence of pronunciations may change.
5. Only *pinyin*, Wade-Giles, and Yale are supported in this release.
6. Except for *Gwoyeu romatzyh*, tone information is not indicated in the romanization. The caller must adjust the returned value according to the tone representation method required.
7. Mandarin pronuciations are based on information from the Unicode database. Only about 25,000 out of 71,000 ideographs in the database have Mandarin pronuciations listed. Many characters which have Mandarin pronunciations not have those pronunciations in the database.
8. For a few characters, the Unicode database lists *pinyin* pronunciations which have no equivalents in either Wade-Giles or Yale romanizations. It is not known at this time the cause of these discrepancies, but the consistency of use in many cases indicates it is probably not simply typographic error. For those pronunciations, the *pinyin* pronunciations have been mapped to Wade-Giles and to Yale as follows:

Table 5-1. Special *pinyin* mappings

pinyin	**Wade-Giles**	**Yale**
dia	*tia*	*tya*
hng	*hng*	*hng*
m	*m*	*m*
n	*n*	*n*
ng	*ng*	*ng*
tei	*t’ei*	*tei*
yo	*yo*	*yo*

5.1.9. `nga_memory_allocate ()`

Allocate dynamic memory.

Synopsis.

```
civ_status_T    nga_memory_allocate
                        ( size_t                amount,
                        void                 ** buffer ) ;
```

Arguments.

`amount`

(in) the amount of memory to allocated (in bytes or characters)

`buffer`

(out) the allocated memory

Returns. Standard exceptions only.

Description. Allocate dynamic memory. This routine is used by all **`nigeria`** procedures to acquire dynamic memory for strings.

Some environments use special-purpose memory management schemes for different kinds of data. By using a common routine for all string allocation, it is possible to convert the library more easily to use such a special-purpose memory management routine for strings.

5.1.10. `nga_memory_free ()`

Release dynamic memory.

Synopsis.

```
civ_status_T    nga_memory_free
                        ( void                * buffer ) ;
```

Arguments.

`buffer`

(in) memory previously acquired by a call to **`nga_memory_allocate ()`**.

Returns. Standard exceptions only.

Description. Release dynamic memory. This routine is used by all **`nigeria`** procedures to release dynamic memory used for strings.

Some environments use special-purpose memory management schemes for different kinds of data. By using a common routine for all string allocation, it is possible to convert the library more easily to use such a special-purpose memory management routine for strings.

5.1.11. `nga_memory_region ()`

Determine the memory region used for dynamic allocation of strings.

Synopsis.

```
civ_status_T    nga_memory_region
                        ( civ_memory_region_T * region ) ;
```

Arguments.

`region`

(out) the region used for dynamic allocation of strings

Returns. Standard exceptions only.

Description. Determine the region of memory used for dynamic allocation of strings. It can be used when interoperable software must know which allocation scheme was used for a memory buffer.

Some environments use special-purpose memory management schemes for different kinds of data. By using a common routine for all string allocation, it is possible to convert the library more easily to use such a special-purpose memory management routine for strings.

5.1.12. `nga_string_break ()`

Search a string for any of a set of characters.

Synopsis.

```
civ_status_T    nga_string_break
                        ( civ_superchar_T     * string,
                        civ_superchar_T       * charlist,
                        civ_superchar_T      ** foundchar ) ;
```

Arguments.

`string`

(in) the **`NULL`**-terminated string to be searched.

`charlist`

(in) a string containing the goal characters

`foundchar`

(out) a pointer to the character in **`string`** found from the list in **`charlist`**.

Returns. Standard exceptions only.

Description. Find the first character from a collection of characters in a string. The **`string`** is searched from its beginning, and each of its characters is compared to the characters in **`charlist`**. when and if a match is found, **`foundchar`** is set to point to the matching character in **`string`**. If no characters from **`string`** are found in **`charlist`**, then **`foundchar`** is set to **`NULL`**.

Notes.

1. This procedure is analagous to the common **`strpbrk ()`** (“string pointer break”) procedure for single-byte characters.

5.1.13. `nga_string_compare ()`

Compare two strings.

Synopsis.

```
civ_status_T nga_string_compare
                        ( civ_superchar_T     * string1,
                        civ_superchar_T       * string2,
                        int                   * result ) ;
```

Arguments.

`string1`

(in) the first of the strings to be compared.

`string2`

(in) the second of the strings to be compared.

`result`

(out) the result of the comparison. **`charlist`**.

Returns. Standard exceptions only.

Description. Two strings are compared from the beginning, character by character. The comparison is based on the code value of the character, not on the collating sequence of any locale. If two characters are reached that are different, then the result is determined by comparing the two dissimilar characters. If the end of one string is reached before the other, then the shorter string is deemed to be “less” than the longer string. Only if the two strings are identical in length and content are they deemed to be “equal”. Upon return:

- if **`string1 < string2`**, then **`* result = -1`**.
- if **`string1 = string2`**, then **`* result = 0`**.
- if **`string1 > string2`**, then **`* result = 1`**.

Notes.

1. This procedure is analagous to the common **`strcmp ()`** (“string compare”) procedure for single-byte characters.
2. **`strcmp ()`** collates in order of code value. Use **`nga_collate_setup()`** and **`nga_collate_decide ()`** for more complex collation sequences.

5.1.14. `nga_string_compare_counted ()`

Compare two counted strings.

Synopsis.

```
civ_status_T    nga_string_compare_counted
                        ( civ_superchar_T     * string1,
                        civ_superchar_T       * string2,
                        unsigned long           limit,
                        int                   * result ) ;
```

Arguments.

`string1`

(in) the first of the strings to be compared.

`string2`

(in) the second of the strings to be compared.

`limit`

(in) the maximum number of characters in each string to be compared.

`result`

(out) the result of the comparison. **`charlist`**.

Returns. Standard exceptions only.

Description. Two strings are compared from the beginning, character by character. At most, **`limit`** characters of each string are compared. The comparison is based on the code value of the character, not on the collating sequence of any locale. If two characters are reached that are different, then the result is determined by comparing the two dissimilar characters. If **`limit`** number of leading characters are the same in both strings, then they are considered equal. Otherwise, if the end of one string is reached before the other, then the shorter string is deemed to be “less” than the longer string. Only if the two strings are identical in length and content, or if they are equal in the first **`limit`** characters, are they deemed to be “equal”. Upon return:

- if **`string1 < string2`**, then **`* result = -1`**.
- if **`string1 = string2`**, then **`* result = 0`**.
- if **`string1 > string2`**, then **`* result = 1`**.

Notes.

1. This procedure is analagous to the common **`strncmp ()`** (“counted string compare”) procedure for single-byte characters.
2. **`strcmp ()`** collates in order of code value. Use **`nga_collate_setup()`** and **`nga_collate_decide ()`** for more complex collation sequences.

5.1.15. `nga_string_complement_span ()`

Search a string for characters which are not in a second string.

Synopsis.

```
civ_status_T    nga_string_complement_span
                        ( civ_superchar_T     * string,
                        civ_superchar_T       * charlist,
                        unsigned long         * count ) ;
```

Arguments.

`string`

(in) the **`NULL`**-terminated string to be searched.

`charlist`

(in) a string containing all characters which are not goal characters

`count`

(out) the number of leading characters from **`string`** which are not in **`charlist`**.

Returns. Standard exceptions only.

Description. Find the first character from a string, which is not in a given collection of characters. Starting with the first character, the characters in **`string`** are examined one by one. If the end of **`string`** is reached, or if a character is found which is in **`charlist`**, then examination stops, and the previous number of characters examined is returned in **`count`**.

Notes.

1. This procedure is analagous to the common **`strcspn ()`** (“string complement span”) procedure for single-byte characters.

5.1.16. `nga_string_concatenate ()`

Append one string to another string.

Synopsis.

```
civ_status_T    nga_string_concatenate
                        ( civ_superchar_T     * destination,
                        civ_superchar_T       * source ) ;
```

Arguments.

`destination`

(in/out) the **`NULL`**-terminated string to which the **`source`** will be appended.

`source`

(in) the **`NULL`**-terminated string which will be appended to the **`destination`** string.

Returns. Standard exceptions only.

Description. Appends the **`source`** string to the **`destination`** string. No new buffer is allocated: it is presumed that the buffer containing the **`destination`** is large enough to hold the characters from both strings together.

The contents of the **`source`** string, with its trailing **`NULL`** character, are copied to the location beginning with (and overwriting) the trailing **`NULL`** of the **`destination`** string.

Notes.

1. This procedure is analagous to the common **`strcat ()`** (“string concatenate”) procedure for single-byte characters.
2. No check is made for the adequacy of the size of the **`destination`** buffer.
3. The strings may not overlap.

5.1.17. `nga_string_concatenate_counted ()`

Append one string to another string.

Synopsis.

```
civ_status_T    nga_string_concatenate_counted
                        ( civ_superchar_T     * destination,
                        civ_superchar_T       * source,
                        unsigned long           limit ) ;
```

Arguments.

`destination`

(in/out) the **`NULL`**-terminated string to which the **`source`** will be appended.

`source`

(in) the string which will be appended to the **`destination`** string.

`limit`

(in) the maximum number of characters to be appended to the **`destination`** string.

Returns. Standard exceptions only.

Description. Appends at most **`limit`** characters of the **`source`** string to the **`destination`** string. No new buffer is allocated: it is presumed that the buffer containing the **`destination`** is large enough to hold the characters from both strings together.

The contents of the **`source`** string, are copied to the location beginning with (and overwriting) the trailing **`NULL`** of the **`destination`** string. Copying stops after a **`NULL`** character is copied, or after **`limit`** characters have been copied, whichever comes first. A trailing **`NULL`** character is always appended to the resulting (combined) **`destination`** string, so the maximum required size of the **`destination`** buffer will be = *(original length of destination) + limit + 1*.

Notes.

1. This procedure is analagous to the common **`strncat ()`** ("string concatenate counted") procedure for single-byte characters.
2. No check is made for the adequacy of the size of the **`destination`** buffer.
3. The strings may not overlap.

5.1.18. `nga_string_copy ()`

Copy a string to a buffer.

Synopsis.

```
civ_status_T    nga_string_copy
                        ( civ_superchar_T     * destination,
                        civ_superchar_T       * source ) ;
```

Arguments.

`destination`

(in/out) the buffer to contain a copy of the **`source`** string.

`source`

(in) the **`NULL`**-terminated string to be copied to the **`destination`** buffer.

Returns. Standard exceptions only.

Description. Copies a string, including its trailing **`NULL`** character, to a buffer. No check is made to ascertain that the **`destination`** buffer is large enough to hold the result.

Notes.

1. This procedure is analagous to the common **`strcpy ()`** (“string copy”) procedure for single-byte characters.

5.1.19. `nga_string_copy_counted ()`

Copy a counted string to a buffer.

Synopsis.

```
civ_status_T    nga_string_copy_counted
                        ( civ_superchar_T     * destination,
                        civ_superchar_T       * source,
                        unsigned long           limit ) ;
```

Arguments.

`destination`

(in/out) the buffer to contain a copy of the **`source`** string.

`source`

(in) the string to be copied to the **`destination`** buffer.

`limit`

(in) the maximum number of characters of the **`source`** string to be copied to the **`destination`** buffer.

Returns. Standard exceptions only.

Description. Copies a string to a buffer. Copying stops after a **`NUL`** character has been copied, or after **`limit`** characters have been copied, whichever comes first. If

there is no **NULL** within the first **limit** characters of the **source** string, then the resulting string will *not* be **NULL**-terminated. No check is made to ascertain that the **destination** buffer is large enough to hold the result.

If the **source** string is shorter than **limit** characters, then **NULL** characters are appended to the **destination** until a total of **limit** characters have been stored.

Notes.

1. This procedure is analagous to the common **strncpy ()** ("counted string copy") procedure for single-byte characters.

5.1.20. nga_string_duplicate ()

Duplicate a string, allocating a new buffer to contain it.

Synopsis.

```
civ_status_T    nga_string_duplicate
                        ( civ_superchar_T     * original,
                        civ_superchar_T      ** duplicate ) ;
```

Arguments.

original

(in) the **NULL**-terminated string to be duplicated.

duplicate

(out) the buffer containing the resulting duplicate string.

Returns. Standard exceptions only.

Description. Allocate a new buffer, and copy a string into the new buffer.

Notes.

1. This procedure is analagous to the common **strdup ()** ("string duplicate") procedure for single-byte characters.

5.1.21. `nga_string_duplicate_counted ()`

Duplicate a counted string, allocating a new buffer to contain it.

Synopsis.

```
civ_status_T    nga_string_duplicated
                        ( civ_superchar_T     * original,
                        unsigned long           limit,
                        civ_superchar_T      ** duplicate ) ;
```

Arguments.

`original`

(in) the string to be duplicated.

`limit`

(in) the maximum number of characters of the **`original`** string to be duplicated.

`duplicate`

(out) the buffer containing the resulting duplicate string.

Returns. Standard exceptions only.

Description. Allocate a new buffer, and copy up to **`limit`** characters of an existing string into the new buffer. A terminating **`NULL`** character is always added, so the maximum size of the resulting **`duplicate`** buffer is *limit + 1*.

Notes.

1. This procedure is analagous to the common **`strndup ()`** (“string duplicate”) procedure for single-byte characters.
2. This procedure always adds a **`NULL`** terminator, while **`nga_string_copy_counted ()`** does not.

5.1.22. `nga_string_length ()`

Determine the length of a string.

Synopsis.

```
civ_status_T    nga_string_length
                        ( civ_superchar_T     * string,
                        unsigned long         * length ) ;
```

Arguments.

`string`

(in) the **`NULL`**-terminated string for which the length is to be determined.

`length`

(out) the number of characters in the string

Returns. Standard exceptions only.

Description. Determines the length of a string by examining each character until the trailing **`NULL`** is found. The returned **`length`** is the number of characters exclusive of the trailing **`NULL`**.

Notes.

1. This procedure is analagous to the common **`strlen ()`** (“string length”) procedure for single-byte characters.

5.1.23. `nga_string_scan_char ()`

Scan a string for a specific character.

Synopsis.

```
civ_status_T    nga_string_scan_char
                        ( civ_superchar_T     * string,
                        civ_superchar_T         goal,
                        civ_superchar_T      ** foundchar ) ;
```

Arguments.

`string`

(in) the **`NULL`**-terminated string to be scanned.

`goal`

(in) the character sought in **`string`**.

`foundchar`

(out) a pointer to the character in **`string`** if found, otherwise **`NULL`**.

Returns. Standard exceptions only.

Description. Search through a string to find the first occurrence of a specified character.

Notes.

1. This procedure is analagous to the common **`strchr ()`** (“string character scan”) procedure for single-byte characters.

5.1.24. `nga_string_scan_char_reverse ()`

Scan a string in reverse for a specific character.

Synopsis.

```
civ_status_T    nga_string_scan_char_reverse
                        ( civ_superchar_T     * string,
                        civ_superchar_T         goal,
                        civ_superchar_T      ** foundchar ) ;
```

Arguments.

`string`

(in) the **`NULL`**-terminated string to be scanned.

`goal`

(in) the character sought in **`string`**.

`foundchar`

(out) a pointer to the character in **`string`** if found, otherwise **`NULL`**.

Returns. Standard exceptions only.

Description. Search through a string from the last character to the first to find the last occurrence of a specified character.

Notes.

1. This procedure is analagous to the common **`strrchr ()`** (“string reverse character scan”) procedure for single-byte characters.

5.1.25. `nga_string_span ()`

Search a string for any of a set of characters.

Synopsis.

```
civ_status_T    nga_string_span
                        ( civ_superchar_T     * string,
                        civ_superchar_T       * charlist,
                        unsigned long         * length ) ;
```

Arguments.

`string`

(in) the **`NULL`**-terminated string to be searched.

`charlist`

(in) a string containing the goal characters

`length`

(out) the number of leading characters in **`string`** which are all in **`charlist`**.

Returns. Standard exceptions only.

Description. Examine the characters of a **`string`** one by one from the beginning, testing each character to see if it is included in a list of characters. Determine the number of such leading characters which are found among the list.

Notes.

1. This procedure is analagous to the common **`strspn ()`** (“string span”) procedure for single-byte characters.

5.1.26. `nga_string_substring ()`

Search a string for a specified substring.

Synopsis.

```
civ_status_T    nga_string_substring
                        ( civ_superchar_T     * string,
                        civ_superchar_T       * substring,
                        civ_superchar_T      ** foundstr ) ;
```

Arguments.

`string`

(in) the **`NULL`**-terminated string to be searched.

`substring`

(in) the substring to be sought within **`string`**.

`foundstr`

(out) a pointer to the first occurrence of **`substring`** within **`string`**, else **`NULL`** if **`substring`** does not occur.

Returns. Standard exceptions only.

Description. Find the first occurrence of a specified substring within a string.

Notes.

1. This procedure is analagous to the common **`strstr ()`** (“string substring”) procedure for single-byte characters.

5.1.27. `nga_unicode_to_big5 ()`

Determine the Big Five equivalent of a Unicode character.

Synopsis.

```
civ_status_T    nga_unicode_to_big5
                        ( civ_superchar_T       unicode_char,
                        unsigned              * big5_row,
                        unsigned              * big5_column ) ;
```

Arguments.

`unicode_char`

(in) the Unicode character to be mapped to Big Five

`big5_row`

(out) the Big Five row

`big5_column`

*(out)*the Big Five column

Returns. Standard exceptions only.

Description. Determine the Big Five equivalent of a Unicode character. If there is no equivalent, then zeroes are returned for the row and column.

Notes.

1. This procedure maps to the Big Five row and column. It does *not* map to any encoded form. Rows are in the range 1 to 94. Columns are in the range 1 to 157.
2. The single byte characters, equivalent to the first 128 Unicode characters, are not mapped by this routine.

5.1.28. nga_unicode_to_jis_x_0208_0212 ()

Determine the JIS X 0208/0212 equivalent of a Unicode character.

Synopsis.

```
civ_status_T    nga_unicode_to_jis_x_0208_0212
                        ( civ_superchar_T       unicode_char,
                        unsigned              * jis_row,
                        unsigned              * jis_column,
                        int                   * jis_x_0212 ) ;
```

Arguments.

`unicode_char`

(in) the Unicode character to be mapped to JIS X 0208/0212

`jis_row`

(out) the JIS row

`jis_column`

*(out)*the JIS column

`jis_x_0212`

*(out)*set to zero for JIS X 0208, or to non-zero for JIS X 0212

Returns. Standard exceptions only.

Description. Determine the JIS X 0208 or JIS X 0212 equivalent of a Unicode character. If there is no equivalent, then zeroes are returned for the row and column.

Notes.

1. This procedure maps to the JIS row and column. It does *not* map to any encoded form. Rows and columns are each in the range 1 to 94.
2. The set of JIS X 0208 characters, and the set of JIS X 0212 characters are disjoint: a given character will not be found in both sets. However, both use the same 94 by 94 *ku ten* (row, cell) scheme to designate specific characters. A given row and column may designate one character in JIS X 0208, and a different character in JIS X 0212.

   The former is considered a base set, and the latter a supplemental set. They are often used together, with escape sequences or similar mechanisms used to specify to which set a given code belongs.

3. The single byte characters, equivalent to the first 128 Unicode characters, are not mapped by this routine.

5.1.29. `nga_unicode_to_ks_c_5601_5657 ()`

Determine the JIS C 5601/5657 equivalent of a Unicode character.

Synopsis.

```
civ_status_T    nga_unicode_to_ks_c_5601_5657
                        ( civ_superchar_T       unicode_char,
                        unsigned              * ks_row,
                        unsigned              * ks_column,
                        int                   * ks_c_5657 ) ;
```

Arguments.

`unicode_char`

(in) the Unicode character to be mapped to KS C 5601/5657

`ks_row`

(out) the KS row

`ks_column`

*(out)*the KS column

`ks_c_5657`

*(out)*set to zero for KS C 5601, or to non-zero for KS C 5657

Returns. Standard exceptions only.

Description. Determine the KS C 5601 or KS C 5657 equivalent of a Unicode character. If there is no equivalent, then zeroes are returned for the row and column.

Notes.

1. This procedure maps to the KS row and column. It does *not* map to any encoded form. Rows and columns are each in the range 1 to 94.
2. The set of KS C 5601 characters, and the set of KS C 5657 characters are disjoint: a given character will not be found in both sets. However, both use the same 94 by 94 (row by cell) scheme to designate specific characters. A given row and column may designate one character in KS C 5601, and a different character in KS C 5657.

   The former is considered a base set, and the latter a supplemental set. They are often used together, with escape sequences or similar mechanisms used to specify to which set a given code belongs.

3. The single byte characters, equivalent to the first 128 Unicode characters, are not mapped by this routine.
4. This version does not map the non-ideograph characters to KS C 5601 or KS C 5657. This means that Hangul characters are not mapped. A future version will correct this.

5.1.30. `nga_xlate_from_unicode_setup ()`

Prepare to translate a string from Unicode to some other character set.

Synopsis.

```
civ_status_T    nga_xlate_from_unicode_setup
                        ( unsigned char       * options,
                        nga_charset_T           charset,
                        nga_status_T          * status ) ;
```

Arguments.

`options`

(in) the string representing the translation options.

`charset`

(in) the character set to which the Unicode string is to be translated.

`status`

(in/out) a status/state structure to be initialized by this procedure

Returns. Standard exceptions only.

Description. Prepares to translate a string from Unicode to some other character set. Appropriate data structures are allocated for the translation, and the **`status`** is initialized accordingly. The **`options`** string is interpreted, and fields in the data structure are set up to correspond to those options.

Notes.

1. This procedure itself does not perform the translation. The translation is done by one or more calls to **`nga_xlate_from_unicode_continue ()`**, which is passed the **`status`** initialized by this procedure.
2. The string input to **`options`** may be constructed by concatenating predefined constants with names **`NGA_OPTION_*`**. If such options conflict with one another, the last conflicting option in the string takes precedence.
3. The **`status`** data structure is overwritten completely by this procedure. Any existing contents are destroyed.
4. This procedure is not implemented in this version of the **`nigeria`** library. A future version will implement this procedure.

5.1.31. `nga_xlate_from_unicode_continue ()`

Translate a sequence of characters from Unicode to some other character set.

Synopsis.

```
civ_status_T    nga_xlate_from_unicode_continue
                        ( nga_status_T        * status,
                        civ_superchar_T       * srcbfr,
                        unsigned long           srclen,
                        unsigned long         * srcuse,
                        civ_superchar_T       * dstbfr,
                        unsigned long           dstlim,
                        unsigned long         * dstlen ) ;
```

Arguments.

`status`

(in) the status structure initialized by a previous call to **`nga_xlate_from_unicode_setup ()`**.

`srcbfr`

(in) the sequence of characters to be translated.

`srclen`

(in) the number of characters in the **`srcbfr`** sequence.

`srcuse`

(out) the number of characters from **`srcbfr`** used in the translation.

`dstbfr`

(in/out) an buffer to contain the results of the translation.

`dstlim`

(in) the maximum number of bytes which can be placed into the destination buffer **`dstbfr`**.

`dstlen`

(out) the number of bytes placed into the destination buffer **`dstbfr`** by the translation process.

Returns. Standard exceptions only.

Description. Translates a sequence of characters from Unicode to another character set. Translation is done according to options and a target character set specified in a previous call to **`nga_xlate_from_unicode_setup ()`**.

The **`status`** structure encapsulates the translation state across multiple invocations of this procedure, allowing the translation of text spanning multiple source buffers into multiple destination buffers. Also, repeated translations may be done according to the same options without again calling **`nga_xlate_from_unicode_setup ()`**, reducing processing time and program complexity.

When this procedure is called for new source text, the caller should set the **`status -> beginning_of_source`** bit. This bit will be cleared by the procedure. The procedure is then called repeatedly until the final source buffer is passed; at that time, the **`status -> end_of_source`** bit should be set. A new text may be translated by setting **`status -> beginning_of_text`** on a subsequent call.

The last time the procedure is called, the caller should set the **`status -> free_state`** bit, to cause the work buffers to be released.

Notes.

1. If the procedure exhausts its input data, then it will set **`status -> source_buffer_exhausted`** before returning to the caller.
2. If the procedure exhausts its output buffer, and still has more to write, then it will set **`status -> destination_buffer_exhausted`** before returning to the caller.
3. The procedure will sometimes look ahead to the next source code value before translating the current source code value. (For example, it may want to ascertain that the following code value does not combine with the current code value to create a single output character.) If any part of the source text is reflected in the translation state on return, then the **`status -> incomplete_source`** bit will be set.

4. If the procedure has read any source data from the buffer which is not reflected in the destination output, then the **`status -> holding_input`** bit will be set. Some algorithms will "hold" such data while waiting for a new output buffer, and others will leave unused data in the source buffer when the output buffer has been exhausted.
5. If the procedure returns with data remaining in the source buffer (that is, if **`srcuse < srclen`**, and if the user subsequently invokes the procedure with more source data, then the remaining unprocessed input characters should be prepended to the new data before passing it to the procedure.
6. If the procedure is invoked with no output buffer (that is, if **`dstbfr`** is **`NULL`**), then the source data will be translated, but the output discarded. However, the length **`dstuse`** will be updated, so the caller may learn the length of buffer which would have been required had an output buffer been specified. The caller may then allocate a buffer of the necessary size, then call the procedure a second time to perform the actual translation. This is less efficient than using a generous estimate of the output length initially, but may be useful in some situations. Invoking the procedure without an output buffer updates the **`status`** structure, so this will not work for inputs spanning multiple buffers unless all such input buffers are passed to the procedure during the buffer size determination.
7. If many source characters do not have equivalents in the destination character set, and if long substitutions (such as character names) are used, then the output may be many times larger than the input.
8. This procedure is not implemented in this version of the **`nigeria`** library. A future version will implement this procedure.

5.1.32. `nga_xlate_to_unicode_setup ()`

Prepare to translate a string to Unicode from some other character set.

Synopsis.

```
civ_status_T    nga_xlate_to_unicode_setup
                        ( unsigned char       * options,
                        nga_charset_T           charset,
```

```
                        nga_status_T          * status ) ;
```

Arguments.

`options`

(in) the string representing the translation options.

`charset`

(in) the character set which is to be translated to Unicode.

`status`

(in/out) a status/state structure to be initialized by this procedure

Returns. Standard exceptions only.

Description. Prepares to translate a string to Unicode from some other character set. Appropriate data structures are allocated for the translation, and the **`status`** is initialized accordingly. The **`options`** string is interpreted, and fields in the data structure are set up to correspond to those options.

Notes.

1. This procedure itself does not perform the translation. The translation is done by one or more calls to **`nga_xlate_to_unicode_continue ()`**, which is passed the **`status`** initialized by this procedure.
2. The string input to **`options`** may be constructed by concatenating predefined constants with names **`NGA_OPTION_*`**. If such options conflict with one another, the last conflicting option in the string takes precedence.
3. The **`status`** data structure is overwritten completely by this procedure. Any existing contents are destroyed.
4. This procedure is not implemented in this version of the **`nigeria`** library. A future version will implement this procedure.

5.1.33. nga_xlate_to_unicode_continue ()

Translate a sequence of characters to Unicode from some other character set.

Synopsis.

```
civ_status_T    nga_xlate_to_unicode_continue
                        ( nga_status_T        * status,
                        unsigned char         * srcbfr,
                        unsigned long           srclen,
                        unsigned long         * srcuse,
                        civ_superchar_T       * dstbfr,
                        unsigned long           dstlim,
                        unsigned long         * dstlen ) ;
```

Arguments.

status

(in) the status structure initialized by a previous call to **nga_xlate_to_unicode_setup ()**.

srcbfr

(in) the sequence of characters to be translated.

srclen

(in) the number of bytes in the **srcbfr** sequence.

srcuse

(out) the number of bytes from **srcbfr** used in the translation.

dstbfr

(in/out) an buffer to contain the results of the translation.

dstlim

(in) the maximum number of characters which can be placed into the destination buffer **dstbfr**.

`dstlen`

(out) the number of characters placed into the destination buffer **`dstbfr`** by the translation process.

Returns. Standard exceptions only.

Description. Translates a sequence of characters to Unicode from another character set. Translation is done according to options and a target character set specified in a previous call to **`nga_xlate_to_unicode_setup ()`**.

The **`status`** structure encapsulates the translation state across multiple invocations of this procedure, allowing the translation of text spanning multiple source buffers into multiple destination buffers. Also, repeated translations may be done according to the same options without again calling **`nga_xlate_to_unicode_setup ()`**, reducing processing time and program complexity.

When this procedure is called for new source text, the caller should set the **`status -> beginning_of_source`** bit. This bit will be cleared by the procedure. The procedure is then called repeatedly until the final source buffer is passed; at that time, the **`status -> end_of_source`** bit should be set. A new text may be translated by setting **`status -> beginning_of_text`** on a subsequent call.

The last time the procedure is called, the caller should set the **`status -> free_state`** bit, to cause the work buffers to be released.

Notes.

1. If the procedure exhausts its input data, then it will set **`status -> source_buffer_exhausted`** before returning to the caller.
2. If the procedure exhausts its output buffer, and still has more to write, then it will set **`status -> destination_buffer_exhausted`** before returning to the caller.
3. The procedure will sometimes look ahead to the next source code value before translating the current source code value. (For example, it may want to ascertain that the following code value does not combine with the current code value to create a single output character.) If any part of the source text is reflected in the translation state on return, then the **`status -> incomplete_source`** bit will be set.

4. If the procedure has read any source data from the buffer which is not reflected in the destination output, then the **`status -> holding_input`** bit will be set. Some algorithms will "hold" such data while waiting for a new output buffer, and others will leave unused data in the source buffer when the output buffer has been exhausted.
5. If the procedure returns with data remaining in the source buffer (that is, if **`srcuse < srclen`**, and if the user subsequently invokes the procedure with more source data, then the remaining unprocessed input characters should be prepended to the new data before passing it to the procedure.
6. If the procedure is invoked with no output buffer (that is, if **`dstbfr`** is **`NULL`**), then the source data will be translated, but the output discarded. However, the length **`dstuse`** will be updated, so the caller may learn the length of buffer which would have been required had an output buffer been specified. The caller may then allocate a buffer of the necessary size, then call the procedure a second time to perform the actual translation. This is less efficient than using a generous estimate of the output length initially, but may be useful in some situations. Invoking the procedure without an output buffer updates the **`status`** structure, so this will not work for inputs spanning multiple buffers unless all such input buffers are passed to the procedure during the buffer size determination.
7. This procedure is not implemented in this version of the **`nigeria`** library. A future version will implement this procedure.

5.2. `ivory coast` API

`ivory coast` is distributed separately. Please refer to the documentation distributed with that package.

Chapter 6. Support, Bugs, and Announcements

6.1. Reporting Errors and Obtaining Support

Because this is free software, it comes with *no warranty* and *no support*. However, the authors (Bionic Buffalo) are interested in producing quality products, and use the software themselves, so they want to see known problems fixed.

Although no response is guaranteed, please send problem reports to **`support@tatanka.com`**. Be sure to say in your e-mail what software (**`nigeria`**) is being used, and which version (**`A.0.0.4`**) is being used.

If you submit fixes, Bionic Buffalo may incorporate them into the main code base. Please indicate whether or not you would like public acknowledgement of your contribution.

6.2. Mailing List for Announcements

There is a mailing list for announcements about updates, problems, errors, and other news related to the **`nigeria`** library. Each time a new version is made available for download, an announcement will be sent to the list. To subscribe, send an e-mail to **`nigeria-announce-request@tatanka.com`**. The message body should contain a single line

```
subscribe
```

A reply will ask you to confirm your subscription.

To remove yourself from the announcement list, send a mail to the same address, with the body consisting of the single line

```
unsubscribe
```

If you have questions, send an e-mail with a single line

```
help
```

in the body.

Information about subscribers (including names and e-mail addresses) will be kept confidential, and will not be used for other purposes.

Chapter 7. Licenses

The **nigeria** library is available from Bionic Buffalo Corporation under two different licenses: the GNU General Public License (GPL), version 2, and an alternate license.

With the GPL, **nigeria** is free: there is no charge for its use. However, the GPL obligates the user to openly publish the source code for any applications derived from **nigeria** (or other GPL'd programs), including applications which call **nigeria** library subroutines. (However, if you do not give or sell the application to anyone else, and use it only yourself, privately, you do not have to publish the source.)

The full text of the GPL, version 2, is printed below. For more information, visit the GNU website at **http://www.gnu.org**.

With the alternate license, there is a small charge, but the requirement to publish source code is waived. See below for more information about how to obtain an alternate license.

7.1. GNU GENERAL PUBLIC LICENSE

7.1.1. Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author’s protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors’ reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone’s free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

7.1.2. Terms and Conditions for Copying, Distribution, and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program”, below, refers to any such program or work,

and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the

Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the

source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute

the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose

distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

7.1.3. How to Apply These Terms to Your New

Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the program's name and an idea of what it does.

Copyright (C) *yyyy name of author*

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) *year name of author*

Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you

`are welcome to redistribute it under certain conditions; type 'show c' for details.`

The hypothetical commands **`show w`** and **`show c`** should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than **`show w`** and **`show c`**; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program Gnomovision (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989

Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

7.2. Alternate License

You automatically receive the GNU General Public License, Version 2, with the **`nigeria`** library. You do *not* automatically receive any other license. In order to receive the alternate license, you must purchase it from Bionic Buffalo Corporation.

If you have the alternate license, you will be able to use the **`nigeria`** library in your own programs without being required to publish your own programs' source code.

Please write to Bionic Buffalo Corporation for information about the alternate license. The e-mail address is for this information is **`sales@tatanka.com`**.

Physical mail may be sent to Bionic Buffalo Corporation, 2533 North Carson Street, Suite 1884, Carson City, Nevada 89706-0147 USA.

About Bionic Buffalo

Bionic Buffalo Corporation (BBC) develops reliable, secure software and systems for real-time, embedded, and high performance environments.

BBC has pioneered CORBA-compliant tools and applications, and deployed the first commercial ORB specifically engineered for embedded environments. BBC CORBA products include high-performance multi-threaded and multi-processor programs, and secure inter-ORB protocols.

The **`nigeria`** library was originally built for use in BBC's OMG IDL compiler (**`france`**), which handles IDL sources in a number of different character sets. The **`france`** IDL compiler can be built in reentrant, multi-threaded form.

Bionic Buffalo also does custom system and software design and development.

They hope you see God in their products (and everywhere else, too!).

For more information, please visit BBC's web site at **`http://www.tatanka.com`**, or send e-mail to **`query@tatanka.com`**.