Glossary

ASCII. ASCII stands for American Standard Code for Information Interchange. It is a way of representing text and the characters that form text in computers. It can be divided into two sections: plain, or lower, ASCII, which includes numbers, punctuation, plain letters without diacritical marks, whitespace characters such as space and tab, and some control characters such as carriage return; and extended ASCII, which includes letters with diacritical marks as well as other special characters.

Before Stata 14, datasets, do-files, ado-files, and other Stata files were encoded using ASCII.

Automation. Automation, formerly known as OLE Automation, is a communication mechanism between Microsoft Windows applications that provides an infrastructure whereby Windows applications can access and manipulate functions and properties implemented in another application. In Stata, an Automation object enables users to directly access Stata macros, scalars, stored results, and dataset information in ways besides the usual log files.

binary 0. Binary 0, also known as the null character, is traditionally used to indicate the end of a string, such as an ASCII or UTF-8 string.

Binary 0 is obtained by using `char(0)` and is sometimes displayed as `\0`. See **[U] 12.4.10 strL variables and binary strings** for more information.

byte. Formally, a byte is eight binary digits (bits), the units used to record computer data. Each byte can also be considered as representing a value from 0 through 255. Do not confuse this with Stata's `byte` variable storage type, which allows values from −127 to 100 to be stored. With regard to strings, all strings are composed of individual characters that are encoded using either one byte or several bytes to represent each character.

For example, in UTF-8, the encoding system used by Stata, byte value 97 encodes "a". Byte values 195 and 161 in sequence encode "á".

characteristics. Characteristics are one form of metadata about a Stata dataset and each of the variables within the dataset. They are typically used in programming situations. For example, the xt commands need to know the name of the panel variable and possibly the time variable. These variable names are stored in characteristics within the dataset. See **[U] 12.8 Characteristics** for an overview and **[P] char** for a technical description.

class. A class is an implementation of object-oriented programming. A class is a set of variables or related functions or both tied together under one name. Stata has two class implementations, one for ado-programming (see **[P] class**) and one for Mata (see **[M-2] class**).

code pages. A code page maps extended ASCII values to a set of characters, typically for a specific language or set of languages. For example, the most commonly used code page is Windows-1252, which maps extended ASCII values to characters used in Western European languages. Code pages are essentially encodings for extended ASCII characters.

code point. A code point is the numerical value or position that represents a single character in a text system such as ASCII or Unicode. The original ASCII encoding system contains only 128 code points and thus can represent only 128 characters. Historically, the 128 additional bytes of extended ASCII have been encoded in many different and inconsistent ways to provide additional sets of 128 code points. The formal Unicode specification has 1,114,112 possible code points, of which roughly 250,000 have been assigned to actual characters. Stata uses UTF-8 encoding for Unicode. Note that the UTF-8–encoded version of a code point does not have the same numeric value as the code point itself.

display column. A display column is the space required to display one typical character in the fixed-width display used by Stata's Results window and Viewer. Some characters are too wide for one display column. Each character is displayed in one or two display columns.

All plain ASCII characters (for example, "M" and "9") and many UTF-8 characters that are not plain ASCII (for example, "é") require the same space when using a fixed-width font. That is to say, they all require a single display column.

Characters from non-Latin alphabets, such as Chinese, Cyrillic, Japanese, and Korean, may require two display columns.

See **[U] 12.4.2.2 Displaying Unicode characters** for more information.

encodings. An encoding is a way of representing a character as a byte or series of bytes. Examples of encoding systems are ASCII and UTF-8. Stata uses UTF-8 encoding.

For more information, see **[U] 12.4.2.3 Encodings**.

extended ASCII. Extended ASCII, also known as higher ASCII, is the byte values 128 to 255, which were not defined as part of the original ASCII specification. Various code pages have been defined over the years to map the extended ASCII byte values to many characters not supported in the original ASCII specification, such as Latin letters with diacritical marks, such as "á" and "Á"; non-Latin alphabets, such as Chinese, Cyrillic, Japanese, and Korean; punctuation marks used in non-English languages, such as "<", complex mathematical symbols such as "±", and more.

Although extended ASCII characters are stored in a single byte in ASCII encoding, UTF-8 stores the same characters in two to four bytes. Because each code page maps the extended ASCII values differently, another distinguishing feature of extended ASCII characters is that their meaning can change across fonts and operating systems.

frames. Frames, also known as data frames, are in-memory areas where datasets are analyzed. Stata can hold multiple datasets in memory, and each dataset is held in a memory area called a frame. A variety of commands exist to manage frames and manipulate the data in them. See [D] **frames**.

global macro. See *local macro and global macro*.

higher ASCII. See *extended ASCII*.

local macro and global macro. A local macro is private, meaning it can be viewed only by the program in which it is defined. A global macro is public, meaning the global macro is available to all programs. See **[U] 18.3.1 Local macros**, **[U] 18.3.2 Global macros**, **[U] 18.3.3 The difference between local and global macros**, and [P] **macro**. Also see *macro, macroname, and macro contents*.

locale. A locale is a code that identifies a community with a certain set of rules for how their language should be written. A locale can refer to something as general as an entire language (for example, "en" for English) or something as specific as a language in a particular country (for example, "en_HK" for English in Hong Kong).

A locale specifies a set of rules that govern how the language should be written. Stata uses locales to determine how certain language-specific operations are carried out. For more information, see **[U] 12.4.2.4 Locales in Unicode**.

looping. Looping is repeatedly executing a piece of code as long as a condition is true. In Stata, `while`, `foreach`, and `forvalues` are all looping commands. See [P] **while**, [P] **foreach**, and [P] **forvalues**. Also see [M-2] **for**, [M-2] **do**, and [M-2] **while**.

lower ASCII. See *plain ASCII*.

macro, macroname, and macro contents. A macro is a string of characters, called the macroname, that stands for another string of characters, called the macro contents. When a macroname is referenced, the macro contents are substituted in place of the macroname. See [U] **18.3 Macros** and [P] **macro**. Also see *local macro and global macro*.

macro expansion. Macro expansion is the process of substituting the macro contents for the macro name. See [P] **macro**.

null-terminator. See *binary 0*.

numlist. A numlist is a list of numbers. That list can be one or more arbitrary numbers or can use certain shorthands to indicate ranges, such as `5/9` to indicate integers 5, 6, 7, 8, and 9. Ranges can be ascending or descending and can include an optional increment or decrement amount, such as `10.5(-2)4.5` to indicate 10.5, 8.5, 6.5, and 4.5. See [U] **11.1.8 numlist** for a list of shorthands to indicate ranges.

object-oriented programming. Object-oriented programming is a programming style where code is based around objects, and those objects may have both data and code methods that can operate on the data associated with an object. An object is constructed, and other objects inherit from or are built on top of that object. For instance, an object could be a point, an object built on top of that could be a line, and objects built on top of that could be polygons. Mata, C++, and Java are examples of programming languages that support object-oriented programming.

OLE Automation. See *Automation*.

plain ASCII. We use plain ASCII as a nontechnical term to refer to what computer programmers call lower ASCII. These are the plain Latin letters "a" to "z" and "A" to "Z"; numbers "0" through "9"; many punctuation marks, such as "!"; simple mathematical symbols, such as "+"; and whitespace and control characters such as space (" "), tab, and carriage return.

Each plain ASCII characters is stored as a single byte with a value between 0 and 127. Another distinguishing feature is that the byte values used to encode plain ASCII characters are the same across different operating systems and are common between ASCII and UTF-8.

Also see *ASCII* and *encodings*.

plugin. A plugin is a piece of software written in another language that adds features to a software package. Stata can call plugins written in C/C++ or Java. Plugins are useful when desired functionality is not available in Stata's ado or Mata languages or for custom methods that require speed and involve heavy looping, recursion, or other computationally demanding approaches. See [P] **plugin**, [P] **Java intro**, and [P] **PyStata integration**.

PyStata. PyStata refers to the integration between Python and Stata. PyStata includes Python integration via the `python` suite of commands, which enables you to call Python from within Stata; the pystata Python package, which allows you to invoke Stata from a standalone Python environment; and the Stata Function Interface module. See [P] **PyStata intro**, [P] **PyStata integration**, and [P] **PyStata module**.

recursion. Recursion is a programming technique where a problem is solved by a function calling itself repeatedly in a nested fashion. Each call is intended to solve a smaller piece of the original problem.

str1, **str2**, . . . , **str2045**. See *strL*.

strL. `strL` is a storage type for string variables. The full list of string storage types is `str1`, `str2`, . . . , `str2045`, and `strL`.

`str1`, `str2`, . . . , `str2045` are fixed-length storage types. If variable `mystr` is `str8`, then 8 bytes

are allocated in each observation to store `mystr`'s value. If you have 2,000 observations, then 16,000 bytes in total are allocated.

Distinguish between storage length and string length. If `myvar` is `str8`, that does not mean the strings are 8 characters long in every observation. The maximum length of strings is 8 characters. Individual observations may have strings of length 0, 1, ..., 8. Even so, every string requires 8 bytes of storage.

You need not concern yourself with the storage length because string variables are automatically promoted. If `myvar` is `str8`, and you changed the contents of `myvar` in the third observation to "Longer than 8", then `myvar` would automatically become `str13`.

If you changed the contents of `myvar` in the third observation to a string longer than 2,045 characters, `myvar` would become `strL`.

`strL` variables are not necessarily longer than 2,045 characters; they can be longer or shorter than 2,045 characters. The real difference is that `strL` variables are stored as varying length. Pretend that `myothervar` is a `strL` and its third observation contains "this". The total memory consumed by the observation would be $64+4+1 = 69$ bytes. There would be 64 bytes of tracking information, 4 bytes for the contents (there are 4 characters), and 1 more byte to terminate the string. If the fifth observation contained a 2,000,000-character string, then $64 + 2{,}000{,}000 + 1 = 2{,}000{,}069$ bytes would be used to store it.

Another difference between `str1`, `str2`, ..., `str2045`, and `strL`s is that the `str#` storage types can store only ASCII strings. `strL` can store ASCII or binary strings. Thus a `strL` variable could contain, for instance, the contents of a Word document or a JPEG image or anything else.

`strL` is pronounce *sturl*.

titlecase, **title-cased string**, and **Unicode title-cased string**. In grammar, titlecase refers to the capitalization of the key words in a phrase. In Stata, titlecase refers to (a) the capitalization of the first letter of each word in a string and (b) the capitalization of each letter after a nonletter character. There is no judgment of the word's importance in the string or whether the letter after a nonletter character is part of the same word. For example, "it's" in titlecase is "It'S".

A title-cased string is any string to which the above rules have been applied. For example, if we used the `strproper()` function with the book title *Zen and the Art of Motorcycle Maintenance*, Stata would return the title-cased string `Zen And The Art Of Motorcycle Maintenance`.

A Unicode title-cased string is a string that has had Unicode title-casing rules applied to Unicode words. This is almost, but not exactly, like capitalizing the first letter of each Unicode word. Like capitalization, title-casing letters is locale-dependent, which means that the same letter might have different titlecase forms in different locales. For example, in some locales, capital letters at the beginning of words are not supposed to have accents on them, even if that capital letter by itself would have an accent.

If you do not have characters beyond plain ASCII and your locale is English, there is no distinction in results. For example, `ustrtitle()` with an English locale locale also would return the title-cased string `Zen And The Art Of Motorcycle Maintenance`.

Use the `ustrtitle()` function to apply the appropriate capitalization rules for your language (locale).

token. A token is a single piece of a text string. Tokens are usually delimited by whitespace or special characters such as commas, brackets, and parentheses.

Unicode. Unicode is a standard for encoding and dealing with text written in almost any conceivable living or dead language. Unicode specifies a set of encoding systems that are designed to hold (and, unlike extended ASCII, to keep separate) characters used in different languages. The Unicode

standard defines not only the characters and encodings for them, but also rules on how to perform various operations on words in a given language (locale), such as capitalization and ordering. The most common Unicode encodings are mUTF-8, UTF-16, and UTF-32. Stata uses UTF-8.

Unicode character. Technically, a Unicode character is any character with a Unicode encoding. Colloquially, we use the term to refer to any character other than the plain ASCII characters.

Unicode normalization. Unicode normalization allows us to use a common representation and therefore compare Unicode strings that appear the same when displayed but could have more than one way of being encoded. This rarely arises in practice, but because it is possible in theory, Stata provides the `ustrnormalize()` function for converting between different normalized forms of the same string.

For example, suppose we wish to search for "ñ" (the lowercase n with a tilde over it from the Spanish alphabet). This letter may have been encoded with the single code point U+00F1. However, the sequence U+006E (the Latin lowercase "n") followed by U+0303 (the tilde) is defined by Unicode to be equivalent to U+00F1. This type of visual identicalness is called canonical equivalence. The one-code-point form is known as the canonical composited form, and the multiple-code-point form is known as the canonical decomposed form. Normalization modifies one or the other string to the opposite canonical equivalent form so that the underlying byte sequences match. If we had strings in a mixture of forms, we would want to use this normalization when sorting or when searching for strings or substrings.

Another form of Unicode normalization allows characters that appear somewhat different to be given the same meaning or interpretation. For example, when sorting or indexing, we may want the code point U+FB00 (the typographic ligature "ff") to match the sequence of two Latin "f" letters encoded as U+0066 U+0066. This is called compatible equivalence.

Unicode title-cased string. See *titlecase, title-cased string, and Unicode title-cased string*.

UTF-8. UTF-8 stands for Universal character set + Transformation Format—8-bit. It is a type of Unicode encoding system that was designed for backward compatibility with ASCII and is used by Stata 14.

version and version control. Version refers to the internal number to which Stata's command interpreter is set. Version control is the process of specifying which version Stata should use for the command interpreter when it processes a command, do-file, or ado-file. For instance, if you write a Stata ado-file and you put `version 15` at the top of your ado-file, then Stata will interpret your ado-file using the syntax that Stata 15 supported even if the version of Stata is now Stata 16 or even Stata 20. Version control is an important feature of Stata because it ensures reproducibility. See [P] **version**.