

Module 10 sample

Codec: encoders and decoders

10.1 What are encoding and decoding?.....	1
10.2 Understanding the Codec algorithms.....	2
10.3 Getting to know Codec	9
10.4 Codec at work	14
10.5 Summary	18
Index	19

Jakarta Codec is an effort to provide implementations for various encoders and decoders that a developer is likely to encounter. At the time of this writing, encoders and decoders are provided for Base64, Hex, phonetic encodings, and URLs. It's possible to extend this API to provide your own implementations for either the supported encodings or any other new encoding scheme.

In this module, we'll look at how to use the Codec component for various encoding/decoding scenarios. We'll start with the basics of each supplied encoding mechanism to help you understand the underlying technology; and we'll show an example of how to implement it using Codec.

10.1 What are encoding and decoding?

If you examine the headers of an email message, a typical header you're likely to encounter is Content-Transfer-Encoding. What is this header used for, and who puts it there? The header is added by the mail client you use to compose the message (in other words, the mail client of the message sender). It's added so that when the mail message passes through the various Internet mail transport mechanisms, it can be identified as having been encoded. This *encoded* message can then pass through these systems, which may have limitations on the data or character sets that they can handle. For example, binary data can't be transferred through these systems if it isn't encoded in a certain way (Base64). This header also identifies to the message recipient the *decoding* to be applied to re-create the original message. This is the basic crux of encoding/decoding. A system encodes a piece of information for safe transmission, conciseness, or security. The receiving system decodes that information based on the encoding used and re-creates the original information. In this section, we'll elaborate on these concepts.

Digital transmission of information requires textual data to be encoded so that differing systems can agree on and understand the information being transmitted. Encoding, however, isn't just required for textual data. Systems require information to be encoded in order to make it concise or unambiguous. Phonetic encoding of related words is an example of such encoding.

The basic premise of encoding is to convert information from one form to another. As we stated earlier, this second form may be relevant for either transmitting this data over channels that are more conducive to the converted form, or it may represent a more concise representation of the original data.

Information that is encoded must be marked with the rules used to do the encoding so that another system can use this marking to understand the encoding rules applied. This other system, can then apply these rules in reverse to arrive at the original information—in effect, decoding the encoded information.

The most prevalent form of digital encoding of textual data, and the one that you may have come across, is the American Standard Code for Information Interchange (ASCII). This encoding method was developed to represent English characters as numbers, with each letter assigned a number from 0 to 127. For example, the ASCII code for uppercase *A* is 65. With 26 letters in the English alphabet and room for more characters like the space and special symbols (such as \$ and %), ASCII is well suited for representing and transmitting textual English information. ASCII also works well to represent Latin-based languages. However, it falls short for other languages, like Arabic, Chinese, and Hindi, which are based on separate scripts and have hundreds of characters that can't be represented by the encoding scheme used in ASCII.

The Unicode encoding scheme (<http://www.unicode.org>) overcomes this problem. It defines three encoding forms that allow the same data to be transmitted in 8-bit, 16-bit, or 32-bit formats. With a theoretical limit of $2^{32} - 1$, Unicode is well suited to represent all the languages of the world.

10.2 Understanding the Codec algorithms

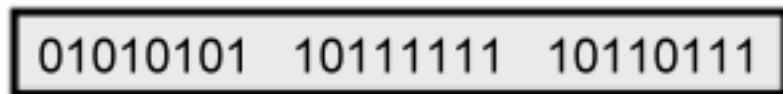
Before we delve into the Codec component, let's look at the algorithms supported by Codec. In the next few sections we'll look at the basics of each encoding scheme supported by Codec.

10.2.1 Base64

An email message that you type and send to a distant recipient is transported across the world through a series of Internet mail gateways that route your message based on header information. This information needs to be in a format that these gateways understand and can interpret for routing forward. However, considering the vast number of gateways around the world and the languages they work in, it's possible that messages would be lost if they weren't sent with headers that contain information in a standard way. It isn't just the headers that need to be in a standard language, though; transporting images and multimedia files requires the conversion of these binary files into transport-safe versions. Toward this end, the Base64 encoding mechanism is defined in rfc2045 (<http://www.ietf.org/rfc/rfc2045.txt>; see module 2 for the definition of an RFC). This RFC deals with the large issues of mail transport and provides other useful information as well. Base64 is only one of the recommendations that is included as part of the Content-Transfer-Encoding header mandate; the other encoding is quoted-printable.

Base64 allows you to convert binary (8-bit) data into a 7-bit short line format. This is necessary because mail transport disallows any other format. For example, you can't transfer mail messages that are binary in format or are represented using 8-bit encoding. This would typically happen if you were transporting media files or images, which, in their natural format, are either binary or 8-bit. Also, by encoding the images and other multimedia files with Base64, you get a textual representation of these files, which may be useful when you want to store the images in an archive.

Base64 works by dividing three continuous octets in the input data into sets of 6 bits each and then representing each 6 bits with the ASCII equivalent. This is better explained by an example. Consider the binary input for some arbitrary data shown in figure 10.1.



```
01010101 10111111 10110111
```

Figure 10.1 Arbitrary binary input

Our task is to convert this binary data into its Base64 equivalent. We start by splitting these 24 bits (3 octets = 24 bits) into groups of 6 bits each to arrive at 4 groups, as shown in figure 10.2.