getRecordIdFromEmail(subject, textBody, htmlBody)

Returns the record ID corresponding to the specified email threading token, or re
is found.

Signature

```
public static Id getRecordIdFromEmail(String subject, String textBody, String
```

Parameters

subject

Type: String

The subject of the email.

textBody

Type: String

The body of the email in text format.

htmlBody

Type: String

The body of the email in HTML format.

Return Value

Type: Id

The record ID that corresponds to the embedded threading token.

Usage

When you send emails with threading tokens embedded in the email subject, th
both the subject and body, most email clients quote the email body and maintai
subject in a response. This method finds a corresponding record that matches th
threading token in a response.

Typically this method is used in Email Services so that you can provide your own l
inbound emails using Apex code.

Example

If you implement header-based threading in your Email Services currently, we rec
use Lightning threading, which combines token-based threading and header-bas
header-based threading to continue to work, store emails as EmailMessage recor
MessagedIdentifier field set properly. With Lightning threading, you can use threa
the primary threading method and rely on header-based threading as a fallback,

```
    global Messaging.InboundEmailResult handleInboundEmail(Messaging.inb
                Messaging.InboundEnvelope env) {

        // Create an InboundEmailResult object for returning the result
        // Apex Email Service.
        Messaging.InboundEmailResult result = new Messaging.InboundEmai

        // Try to find the Case ID using threading tokens in email attr
        Id caseId = EmailMessages.getRecordIdFromEmail(email.subject, em

        // If we haven't found the Case ID, try finding it using headers
        if (caseId == null) {
            caseId = Cases.getCaseIdFromEmailHeaders(email.headers);
        }

        // If a Case isn’t found, create a new Case record.
        if (caseId == null) {
            Case c = new Case(Subject = email.subject);
            insert c;
            System.debug('New Case Object: ' + c);
            caseId = c.Id;
        }

        // Process recipients
        String toAddresses;
        if (email.toAddresses != null) {
            toAddresses = String.join(email.toAddresses, '; ');
        }

        // To store an EmailMessage for threading, you need at minimum
        // the Status, the MessageIdentifier, and the ParentId fields.
        EmailMessage em = new EmailMessage(
            Status = '0',
            MessageIdentifier = email.messageId,
            ParentId = caseId,
            // Other important fields.
            FromAddress = email.fromAddress,
            FromName = email.fromName,
            ToAddress = toAddresses,
            TextBody = email.plainTextBody,
            HtmlBody = email.htmlBody,
            Subject = email.subject,
            // Parse thread-index header to remain consistent with Emai
            ClientThreadIdentifier = getClientThreadIdentifier(email.hea
            // Other fields you wish to add.
        );

        // Insert the new EmailMessage.
        insert em;
        System.debug('New EmailMessage Object: ' + em );

    // Set the result to true. No need to send an email back to the user
    // with an error message.
    result.success = true;

    // Return the result for the Apex Email Service.
    return result;
  }

  private String getClientThreadIdentifier(List<Messaging.InboundEmail.H
    if (headers == null || headers.size() == 0) return null;
    try {
        for (Messaging.InboundEmail.Header header : headers) {
            if (header.name.equalsIgnoreCase('thread-index')) {
                Blob threadIndex = EncodingUtil.base64Decode(header.valu
                return EncodingUtil.convertToHex(threadIndex).substring(
            }
        }
    } catch (Exception e){
        return null;
    }
    return null;
```

DID THIS ARTICLE SOLVE YOUR ISSUE?
Let us know so we can improve!

Sh