**ASCII Text Dump**

[saga](https://www.sqlservercentral.com/author/saga), 2021-01-15 (first published: 2019-11-12)

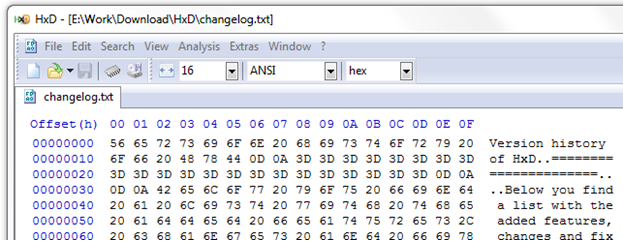
FROM: <https://www.sqlservercentral.com/articles/ascii-text-dump>

Introduction

Recently, I had the task of analyzing a varchar field for special characters. The field contained text entered by the user with a standard US keyboard; however, the proprietary software was breaking up the text and delimiting it with characters whose ASCII values were above 127.

Eyeballing the text that I dumped using a SELECT statement in SQL Server Management Studio (SSMS) revealed the special characters, but I wanted a more formal way of viewing the text plus also their ASCII equivalents. I recalled the Olden Days when I used DEBUG to dump memory contents, among other more modern equivalents to this DOS-era tool.

What I wanted was to display the ASCII values on one side and the original text on the opposing side, much like DEBUG did. The next image presents a partial screen cap of the hex editor HxD which illustrates what I wanted.



This article presents a script with similar functionality.

Design Decisions

My first decision was whether to have this functionality implemented as a stored procedure or a function. The conclusion was simple enough, given my requirements:

1. Pass a generic varchar string to the procedure
2. Use PRINT to display the dump in the messages tab of SSMS
3. Usage of this procedure would be mostly within SSMS

The second item was a deal breaker in possibly using a function because functions do not allow use of the PRINT statement.  I decided that the procedure would PRINT out the dump, instead of returning it as a row as most stored procedures do. After all, my use of this procedure would be exclusively within SSMS.

The next decision was determining what data I would display and in what format. I used the image above as a guide, but decided that I didn’t want to display the offset because for me it wasn’t critical for my analysis. Also, I was tempted to use hexadecimal notation to save one character space per byte, but I found it more practical to display a decimal equivalent of the ASCII value. If it was an “A”, I wanted to see 065.

Implementation

I began my research to determine how feasible this project was. I found various examples of how to get a char’s ASCII value and how to traverse a string. Given the research, I knew that such an endeavor was possible and began developing the algorithm that would accomplish this. I called the stored procedure sp\_Text\_ASCII\_Dump, but it can be any name.

Following is the code for the script used to create the stored procedure.

USE [Dev\_db]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[sp\_Text\_ASCII\_Dump] Script Date: 09/13/2019 13:41:49 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

/\*\*\*\*\*

Name: sp\_Text\_ASCII\_Dump

Author: Salvador Garcia

Date: Sep. 13, 2019

Type: SQL server stored procedure for internal analysis use.

Purpose: Dump the ASCII values of the string that is passed in.

\*\*\*\*\*\*/

create PROCEDURE [dbo].[sp\_Text\_ASCII\_Dump]

@sInString varchar(8000)

AS

BEGIN

--SET NOCOUNT ON added to prevent extra result sets from interfering with SELECT statements.

SET NOCOUNT ON;

declare @sOutString as varchar(1000)

declare @iCnt as int

set @sOutString = ''

set @iCnt = 0

while @iCnt < len(@sInString)

begin

set @iCnt += 1

set @sOutString += RIGHT('000'+CAST(ASCII(substring(@sInString, @iCnt, 1)) AS VARCHAR(3)),3) + ' '

if @iCnt % 16 = 0

begin

print @sOutString + ' ' + substring(@sInString, @iCnt - 15, 16)

set @sOutString = ''

end

end

print @sOutString + space(64 - len(@sOutString)) + substring(@sInString, @iCnt - (@iCnt % 16 - 1), 16)

END

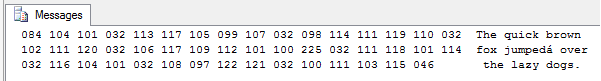
GO

Usage

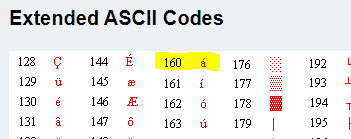
In SSMS, pass a text field to the stored procedure:

exec sp\_Text\_ASCII\_Dump 'The quick brown fox jumpedá over the lazy dogs.'

The result is the following:



The result looked good. I noticed that the ASCII value for “á” was 225 instead of the expected 160, as the next image shows.



This is the “extended ASCII” as defined by IBM for the IBM PC. Since the real ASCII is only 7 bits, its values go from 0 to 127. It wasn’t long before different vendors saw that 128 values were being wasted, since the original ASCII values only used half of the 8-bit range. They saw fit to define special characters and assign them values. The small portion of the chart shown in the image above present but one of these implementations.

It seemed that SQL Server was using a different definition. I did some research, but I was not happy with what I found, so I decided to ask over at the MSDN SQL Server T-SQL forum. Three gentlemen, Erland Sommarskog, Dan Guzman, and Ronen Ariely chimed in.

First, the replies provided some terminology:

* **Character** – This is the symbol as generated by the keyboard. This includes letters, numbers and punctuation marks.
* **Character set** – This is a congruent group of characters, such as English letters and numbers. Some character sets are ASCII, UNICODE and EBCDIC.
* **Code page** – This is an association of a character with a numbering system. Code pages are usually referenced using a number and are supersets of the original 128 character ASCII.
* **Code point** – This is the number that is assigned to the character within a code page. In the above example, the code point 65 represents “A”.
* **Character encoding** – This is the process or algorithm used to create the code points for a code page. Examples of encoding schemes are UTF-8 and UTF-16 (among others).

What they mentioned is that SQL Server manages multiple code pages where each code page is an implementation of how characters are encoded. For the most part, the first 128 values are encoded the same, but the upper 128 values may be encoded differently for different character sets.

To see what code page I was using I went back to SSMS and following some examples presented in the forum’s replies I entered the following in the query window:

SELECT SERVERPROPERTY('Collation')

The result I got was the following:

SQL\_Latin1\_General\_CP1\_CI\_AS

To see the code point for “á” for the current code page I entered the following:

SELECT ASCII((SELECT 'á' COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS))

The result was the following: 225

Yup, there it is. Another example in the forum replies used a different code page:

SELECT ASCII((SELECT 'á' COLLATE SQL\_Latin1\_General\_CP437\_CI\_AS))

The result for that statement was the following: 160

This was the expected value. I just wanted to make note of this so that anyone venturing into this understands that the upper 128 values of the ASCII encoding scheme may have different code points for the characters used. Each code page has a name, or more specifically, a number. The first example above (CP1) uses code page 1252 while the second (CP437) uses code page 437.

Many thanks to the three gents that helped me out with this and in general to everyone who takes time to help out on the forums.

[](https://forms.office.com/Pages/ResponsePage.aspx?id=PCsO473TL0i84iM4oQ4dpAN96d6AjC1AremVW5hZgmpUMFJXOTBTQkFQRDJZQlQ5MkxMVDlTTElZWi4u)

Read 5,172 times  
(860 in last 30 days)

Rate

★ ★ ★ ★ ★ ★ ★ ★ ★ ★

★ ★ ★ ★ ★

4.78 (9)

Share

Save

[Add to briefcase](https://www.sqlservercentral.com/articles/ascii-text-dump#add-to-briefcase)

Categories

* [T-SQL](https://www.sqlservercentral.com/categories/t-sql)

[**Join the discussion and add your comment**](https://www.sqlservercentral.com/forums/topic/ascii-text-dump)

Recent comments

* [Just my 2 cents.  Normally, I'm looking for characters out of the ordinary.  I also normally hav...](https://www.sqlservercentral.com/forums/topic/ascii-text-dump#post-3697058)

[Jeff Moden](https://www.sqlservercentral.com/author/jeff-moden)

* [Suppose you have column ID in table T. Try the following: Select ID, CAST(ID as Varbinary(255)) as ...](https://www.sqlservercentral.com/forums/topic/ascii-text-dump#post-3696953)

[steve.casey](https://www.sqlservercentral.com/author/stevecasey)