Team Developer 5.1 and Unicode

Unify Corporation

Table of Contents

Introduction

Unicode is supported throughout Team Developer 5.1, from string variables to literals, parameters and any object name in the outline.

With this, some changes were introduced:

- All SAL (Scalable Application Language) strings are now able to store Unicode data and any literal string in the outline will be treated as Unicode, using UTF-16 as default encoding.
- Internal Team Developer 5.1 functions which manipulate strings have been enhanced to handle Unicode characters.
- External Function calls and database connectivity will also support Unicode data.
  Refer to section "Database Connectivity known Unicode issues" at the end of this document.
- Team Developer 5.1 runtime DLL (cdlli51.dll) defines both wide ('W') and ANSI ('A') functions to be used with SAL functions that use strings as parameters.

String conversion from ANSI to Unicode and vice-versa occurs automatically from a Team Developer 5.1 application but because of some differences introduced by Unicode support, a few changes on current applications might be necessary and are covered under the sections below.

Outline conversion to Team Developer 5.1

All included libraries (APLs) provided by Team Developer 5.1 that manipulate strings have been converted to use the wide ('W') versions of the APIs instead of the ANSI ('A') ones. For this reason, the first step when migrating an application to Team Developer 5.1 is to make sure that APLs from previous Team Developer versions are not in the application path and will not be used by the application about to be migrated. If so, this will lead to conversion problems.

You can either use Team Developer's Migration Wizard to assist with the conversion process or directly load the application to be migrated into Team Developer's 5.1 IDE. If using the IDE directly, the included files need to be in the path.

When migrating with the wizard or directly with the IDE you need to do it on a machine that has the same local code page that the application was written on. For example if it is a Japanese multi-byte application, the machine that is migrated to TD5.1 on must have the same default code page as the original. Once migrated, the application can be used with any default code page as long as a Unicode font is enabled for the application.

If in the past your files have directly referenced Team Developer's DLLs then you will need to change them to the new versions. This change may be done in the externals section or for a custom control. Easiest way to find these is to save the new file using "Text (*.apt)" format. Open that APT using a Unicode enabled text editor like Windows Notepad. Search for references to old Team Developer DLLs. For instance, if you are migrating from Team Developer 4.2, search for "42.dll" and replace it by "51.dll".

In most cases, Team Developer's outline automatic conversion process will take care of all DLL references but there are situations where this is not possible. So make sure to complete this step as old DLL references may cause unpredictable results, including application crashes.

Migrating applications that call External Functions

Many applications use Team Developer's ability to call external functions from Windows APIs or other third-party DLLs. Even TD's built-in APLs make use of external function calls and as mentioned before they have been already converted to deal with Unicode data. The same step must be accomplished from your application and answering the following questions will determine the necessary actions to be taken:

- Does your application call Windows APIs?
- Does your application call third-party DLLs?
    - If so, is there a Unicode enabled version of that DLL or the source code is available?
- Does your application or any included DLL make use of Team Developer's runtime? (cdllixx.dll)
  "xx" represents Team Developer's version number.

For all cases above due to a change in Team Developer's 5.1 C Runtime library, any external function call must now match the exact number of parameters of its prototype, as well as proper data type must be assigned from Team Developer. In previous versions, TD would accept omitting certain parameters in the external function call and this would not be a problem. Team Developer 5.1 will not accept that discrepancy and a stack problem will occur, so make sure to check and fix any external calls not meeting those criteria.

For better application performance, Unify recommends that all external calls that manipulate strings are changed to the 'W' version of the API in order to use wide characters instead. The reason behind this is that Team Developer will automatically convert from ANSI to Unicode and reverse every time it determines that LPSTR, LPCSTR or CHAR is used as parameters in external function calls. Pre-converting your application to always use wide characters will avoid this extra step.

Note: that step is not mandatory unless multiple languages are used in the same application and the application calls external functions. Since Team Developer's automatic conversion occurs for the current local page only, mixing multiple language characters and not using 'W' versions of the APIs will cause only characters recognized by your current local page to be converted, causing question marks ('?") to show up for non-current local page characters. Again, pre-converting external calls to always use wide characters will avoid Team Developer's automatic conversion and consequently this problem.

As mentioned before, Team Developer 5.1 will automatically convert LPSTR, LPCSTR and CHAR parameters from Unicode to ANSI when passing strings to external functions and from ANSI to Unicode when receiving strings from external functions. But keep in mind that received strings are now double the size of what they used to be in previous TD versions due to UTF-16 default encoding (see notes below).

Team Developer programmers were always responsible for setting the appropriate receive buffer size by calling SalStrSetBufferLength( ).

In TD5.1 base release, SalStrSetBufferLength( ) doubles the buffer length passed as parameter. It was implemented this way so developers wouldn't have to change their code when migrating legacy applications. This ended up not being the best approach in some situations so in TD5.1 Service Pack 2 Unify introduced two new functions:

1. **SalSetBufferLength (sTargetStr, nBuffLength)**

2. **SalGetBufferLength (sTargetStr)**

The new function SalSetBufferLength() doesn't double the length passed as parameter. So it's the developer's responsibility to handle the buffer size when dealing with Unicode data.

We recommend that developers use those two new functions from Service Pack 2 and on. The old SalStrSet/GetBufferLength( ) equivalents will be deprecated in the near future. Please see TD's help for details on these functions.

Note 1: generally, each character in the UTF-16 encoding will use 2 bytes plus the Null terminator which also uses 2 bytes. Take this into consideration when checking the buffer size of received strings in Team Developer 5.1.

Note 2: be aware of an important difference between SAL functions SalStrLength( ) and SalGetBufferLength( ). The first will return the number of characters in a string while the latter will return the buffer length of a

string in bytes. In this case, the length also includes the string's Null terminator character.

Applications calling third-party DLLs

If your application calls a third-party DLL and a Unicode version of that DLL is available, you should consider changing your Team Developer 5.1 application code to call it instead, for the reasons explained before. If not and you don't have access to the DLL source code, you can still trust on Team Developer's automatic conversion of strings but you'll need to check for the other issues described in this document, like buffer size manipulation of received strings and matching the exact number of parameters of the prototype function as well as assigning the proper data type from Team Developer.

Applications calling Team Developer's runtime

If your application or any included DLL make use of Team Developer's runtime they have to be recompiled to use the new "cdlli51.dll". If you don't have access to the included DLL source code, make sure that the DLL vendor provides you one or a 5.1 recompiled version of it. If you have access to the source code, it's now time to decide if you want to convert it from ANSI to Unicode or not.

Converting to Unicode is not mandatory as "cdlli51.dll" provides both ASCII and Unicode versions of SAL functions that handle strings, so no change in the source code calls are necessary. The one change that must be done is to define *TDASCII* in the project to force it to compile using the ASCII version of the SAL function instead of the Unicode one. This is explained in details in the "Team Developer ASCII API.doc" available after installing Team Developer 5.1 samples. Its reading is strongly recommended. Download Team Developer 5.1 samples here:
http://www.unify.com/Services/productDownloads.aspx

If you're using HSTRING parameters when exchanging data with Team Developer's runtime please be advised of the following difference:

Because Team Developer 5.1 is Unicode enabled, all string data originating from the TD runtime is going to be encoded in UTF-16 instead of ASCII. If a user's DLL uses the HSTRING data type to manipulate the string buffer in the DLL and assumes ASCII, it will need to be trans-coded before it can be used. The opposite is also true: an HSTRING has to be converted from ASCII to UTF-16 when passed back to the Team Developer runtime.

Because HSTRING has been used by programmers to handle data other than strings only, it is not converted automatically when migrating an application from a previous version to TD5.1 as it's the case with other data types like LPSTR, LPCSTR and CHAR. The conversion has to be done manually by calling the new functions below:

SalStrToMultiByte() should be called to change the encoding of a string from UTF-16 to the encoding expected by the DLL's code.

SalStrToWideChar() should be called to switch the encoding back to UTF-16 as expected by Team Developer.

Usage:

1. **SalStrToMultiByte (sSource, sTarget, nEncoding)**
   - sSource: HSTRING input parameter
   - sTarget: LPHSTRING trans-coded output
   - nEncoding: any valid code page identifier available in the system
2. **SalStrToWideChar(sSource, sTarget, nEncoding)**
   - sSource: HSTRING input parameter
   - sTarget: LPHSTRING trans-coded output
   - nEncoding: any valid code page identifier available in the system

Please refer to the "SalASCII_dll.app" sample available under Team Developer's installation directory and its respective "Team Developer ASCII API" document for other Unicode-related information.

Database Connectivity known Unicode issues

- When inserting Unicode string constants from a Team Developer 5.1 application into Microsoft SQL Server, make sure to precede them with a capital letter N. Otherwise, SQL Server will convert them to the non-Unicode code page of the current database when inserting.
- When dealing with Unicode strings and Oracle databases, modify the database character set to UTF-8 by editing NLS_LANG settings.

- This is not necessary if using TD5.1 latest Service Pack.

- UTF-8 encoding must also be set for Sybase databases by editing the file "local.dat" under \Sybase\local sub-directory when dealing with Unicode strings.

  - This is not necessary if using TD5.1 latest Service Pack.

- When connecting to SQLBase International or SQLBase 11 from Team Developer 5.1 there is no extra setting required for Unicode.

- If mixing different languages into a single database, Unicode must be enabled for that database in order to show characters correctly. If Unicode is not enabled, question marks ('?') will show up in Team Developer 5.1 applications.

For databases other than Unify SQLBase, please contact the specific vendor for details on enabling Unicode and configuring the settings mentioned above.

About Unify

Unify Corporation
2101 Arena Blvd., Suite 100
Sacramento, CA 95834
USA
Phone: 1.916.928.6400
Toll Free: 1.800.468.6439
Fax: 1.916.928.6404
United Kingdom: +44 (0)1753 245 510
France: +33 (0)1 34 58 28 30