Programmer's Reference

ApSIC Xbench Programmer's Reference for QA Plugins

3.0

Printed on 19 March, 2015

Copyright Notice

ApSIC, S.L.
Caballero, 76 4-3
08029 Barcelona
Spain
Website: http://www.xbench.net

Contents

Copyright Notice 2

Implementation Guidelines 3

QA plug-in Structure 5
  Startup Stage....5
  QA Check Stage 7

Installation 11

Reference 13

Functions 15
  GetDeclareInfo 15
  GetDeclareName 15
  GetFirstFunction....16
  GetFirstResult....16
  GetFunctionName....17
  GetNextFunction 18
  GetNextResult 19
  ProcessBegin 20
  ProcessEnd 21
  ProcessSegment....22
  SetConfigFile....23
Structures 25
  TQAPluginDeclareInfo 25
  TQAPluginResult 26
  TQAPluginResultOptions....27
  TQASegmentInfo 28

Implementation Guidelines

An ApSIC Xbench QA plug-in is a standard Windows DLL that implements a predefined set of function calls designed to extend the QA built-in capabilities.

The QA plug-in can be implemented in any programming language that supports standard Windows DLL development.

All function calls described in this reference guide must be implemented and exported. If not all functions are exported, ApSIC Xbench will either not recognize the QA plug-in or crash.

The name of all ApSIC Xbench QA plug-in DLLs must begin with *XbenchQA*.

CHAPTER 1

QA plug-in Structure

The QA plug-in implements two sets of functions that are used by ApSIC Xbench at two different stages:

Startup Stage

These functions are called when ApSIC Xbench starts and are used to get information about the functionality of all QA plug-ins installed.

ApSIC Xbench uses this information to extend the check box options available in the QA view. Each QA plug-in maps to an item in the *check group* list and each check function implemented in the QA plug-in maps to a check in the *list of checks* list linked to the check group. All check functions are enabled by default.

The function calls used at this stage are:

For every QA plug-in:

- ***GetDeclareName***
- ***GetDeclareInfo***
- ***GetFirstFunction***
- ***SetConfigFile***

For every check function in the QA plug-in:

- ***GetFunctionName***
- ***GetNextFunction***

The following flowchart shows how ApSIC Xbench uses these functions upon startup:

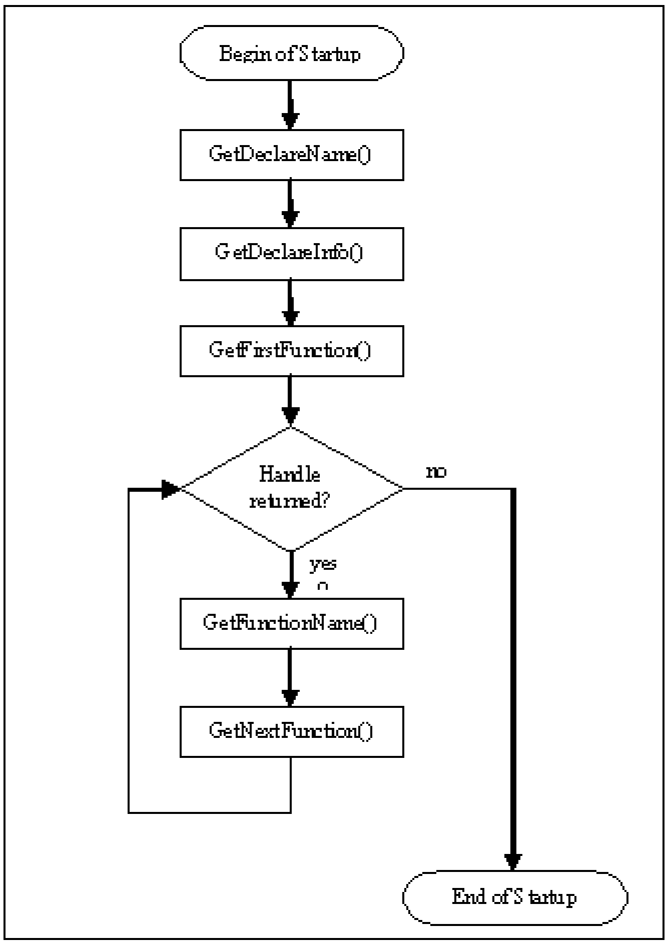

Figure 1 Startup Flowchart

QA Check Stage

These functions are called in the QA check process started from ApSIC Xbench using the **Check Ongoing Translation** button.

The QA process works at two different levels: segment level and process level. The segment level is used to check a segment without context information and the process level is used to check for QA issues that depend on the full set of segments included in the QA check. Every check function in the QA plug-in can implement either one level or both.

Examples of QA checks at segment level are: repeated words, number formats and spell checks. Examples of QA checks at process level are: translation inconsistencies and cross-reference issues.

The function calls used at this stage are:

- ***ProcessBegin***
- ***ProcessEnd***
- ***ProcessSegment***
- ***GetFirstResult***
- ***GetNextResult***

The following flowchart shows how ApSIC Xbench uses these functions upon a QA check start:

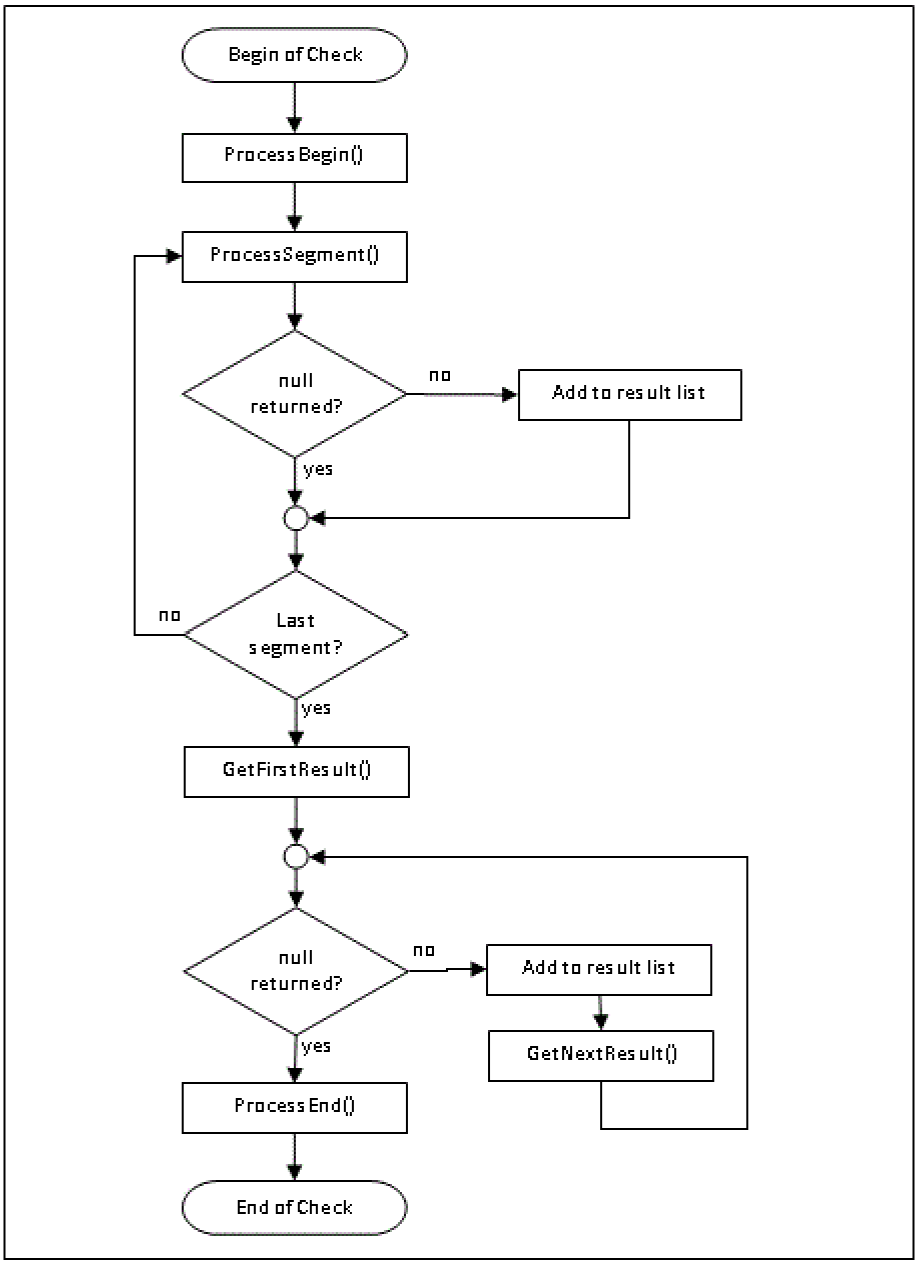

Figure 2 QA Check Flowchart

The ProcessBegin call is used to notify the QA plug-in that a new check function has been launched and the QA plug-in should initialize all internal structures used by this check function.

Then, ApSIC Xbench calls ProcessSegment for each segment included in the QA segment set, providing the QA plug-in with the segment data needed. This function should implement all segment level checks and return a pointer to a ***TQAPluginResult*** structure with error information or NULL if nothing to report is detected.

If a process level check is needed, this function call should also store the segment data received as a parameter in an internal working structure.

When ProcessSegment has been called for every segment, ApSIC Xbench calls GetFirstResult. If no process check is implemented, this function must simply return NULL, otherwise it should perform the process checks using the segment data stored in the previous ProcessSegment calls. When checks are finished, it should return a pointer to a TQAPluginResult with the information of the first issue found.

ApSIC Xbench will add this issue in the QA result list and will keep calling GetNextResult function to retrieve subsequent errors until NULL is returned.

At the end of the process, ProcessEnd will be called to inform the QA plug-in that all working resources allocated during the QA checks should be freed.

Installation

At startup, ApSIC Xbench will look for all DLLs in the ApSIC Xbench directory with a file name beginning with *XbenchQA*. Those DLLs implementing all function calls defined in this reference guide will be loaded and prepared to be used by the QA process.

To install a QA plug-in, just copy the DLL into the ApSIC Xbench directory. To uninstall it, simply move the DLL to another directory.

Reference

CHAPTER 2

Functions

GetDeclareInfo

Retrieves information about the QA plug-in.

Syntax

```
PQAPluginDeclareInfo GetDeclareInfo()
```

Parameters

None

Result

Returns a pointer to a ***TQAPluginDeclareInfo*** object containing information about the QA plug-in.

Comments

GetDeclareName

Retrieves the name for this QA plug-in to be shown in the “Check Group” box in QA tab.

Syntax

```
PWSTR GetDeclareName()
```

Parameters

None

Result

Returns a pointer to a string with the name of the QA plug-in.

Comments

GetFirstFunction

Retrieves the handle of the first function in the QA plug-in.

Syntax

```
DWORD GetFirstFunction()
```

Parameters

None

Result

Returns a handle for the first QA plug-in function. This handle will be later used to refer to this specific function during the QA pass.

Comments

GetFirstResult

Notifies the function QA plug-in that no more segments will be processed so it can perform any postprocess if needed, and returns the first of the errors that have to be notified in the QA tab.

Syntax

```
PQAPluginResult GetFirstResult(
  DWORD aHandle
)
```

Parameters

aHandle: Function handle as obtained with GetFirstFunction and GetNextFunction.

Result

Returns a pointer to a TQAPluginResult structure if an error has to be notified or NULL if there are not process errors left.

Comments

This function involves two actions:

a) it tells the QA plug-in that ProcessSegment will not be called again meaning that all segments have been checked, and

b) it returns the first of the errors found or NULL if none is found or if the errors have been notified in ProcessSegment.

GetFunctionName

Retrieves the name of a specific function of the QA plug-in.

Syntax

```
PSTR GetFunctionName(
  DWORD aHandle
)
```

Parameters

aHandle: Function handle as obtained with GetFirstFunction and GetNextFunction.

Result

Returns a string with the function name assigned to *aHandle*. ApSIC Xbench will display this string in the “List of Checks” box in the QA tab.

Comments

GetNextFunction

Retrieves the handle of the next function contained in the QA plug-in.

Syntax

```
DWORD GetNextFunction()
```

Parameters

None

Result

Returns a handle for the next QA plug-in function or zero if no more functions are present.

Comments

After calling GetFirstFunction to obtain the handle for the first function, ApSIC Xbench will keep calling GetNextFunction until the result value is 0.

GetNextResult

Retrieves the next error that must be reported on the QA tab.

Syntax

```
PQAPluginResult GetNextResult(
  DWORD aHandle
)
```

Parameters

aHandle: Function handle as obtained with GetFirstFunction and GetNextFunction.

Result

Returns a pointer to a TQAPluginResult structure if an error has to be notified or NULL if there are no errors left.

Comments

If the call to GetFirstResult does not return NULL, ApSIC Xbench will keep calling GetNextResult until NULL is returned.

ProcessBegin

Notifies the function that a new QA process is being launched and it should initialize.

Syntax

```
  void ProcessBegin(
    DWORD aHandle, PQAFunctionParams aParams
  )
```

Parameters

aHandle: Function handle as obtained with GetFirstFunction and GetNextFunction.

aParams: (currently ignored) Optional parameters to be used by the function. Should be NULL.

Result

None

Comments

ApSIC Xbench will call this function for every function QA plug-in that will be checked, but not for those that the user has manually disabled in the QA tab, so it should NOT be assumed that ProcessBegin has been called for other functions in the same QA plug-in.

ProcessEnd

Notifies the function that the QA process has finished and all allocated resources should be freed.

Syntax

```
void ProcessEnd(
  DWORD aHandle
)
```

Parameters

aHandle: Function handle as obtained with GetFirstFunction and GetNextFunction.

Result

None

Comments

ProcessSegment

Provides the function QA plug-in with a segment to be checked or stored in working structures.

Syntax

```
PQAPluginResult ProcessSegment(
   DWORD aHandle, TQASegmentInfo aSegInfo
 )
```

Parameters

aHandle: Function handle as obtained with GetFirstFunction and GetNextFunction.

aSegInfo: Structure with information about the segment to be checked.

Result

Returns a pointer to a TQAPluginResult structure if an error has been found in this segment or NULL if nothing has to be notified.

Comments

There are two ways of notifying ApSIC Xbench about errors in segments: you can either return the value in ProcessSegment or use GetFirstResult and GetNextResult to return the list of errors at the end of the QA check. If you want to use the later method you can simply return NULL as the result for ProcessSegment.

The *Options* field of TQAPluginResult structure will be ignored and can be NULL.

SetConfigFile

Sets a filename for the QA plug-in to be used as a configuration file.

Syntax

```
void SetConfigFile(
  PSTR aFile
)
```

Parameters

aFile: Filename to be used by the QA plug-in as a configuration file.

Result

None

Comments

This function allows ApSIC Xbench to provide a filename that the QA plug-in may use for storing and retrieving some configuration values, so the QA plug-in does not need to use a hardcoded filename or guess an appropriate file path.

Currently ApSIC Xbench does not call this function but it must be exported by the QA plug-in anyway.

CHAPTER 3

Structures

TQAPluginDeclareInfo

Contains information and properties regarding the QA plug-in.

Syntax

```
 typedef struct _TQAPluginDeclareInfo {
    WORD StructVersion;
    WCHAR Description[256];
    WCHAR Keywords[256];
    WORD Version;
    LANGID SourceLang;
    LANGID TargetLang;
    BOOL AllowsUnicode;
    BOOL AllowsAnsi;
  } TQAPluginDeclareInfo, *PQAPluginDeclareInfo;
```

Properties

StructVersion: Version of the *TQAPluginDeclareInfo* structure. Leave as 0.

Description: Short description of the QA plug-in functionality.

Keywords: Comma separated list of keywords which are related to this QA plug-in (not used currently).

Version: Version for this QA plug-in.

SourceLang: Language ID for the source language accepted by the QA plug-in. Leave as 0 if any (not checked currently).

TargetLang: Language ID for the target language accepted by the QA plug-in. Leave as 0 if any (not checked currently).

AllowsUnicode: Indicates if the QA plug-in is able to handle Unicode strings natively so they will be used if available.

AllowsAnsi: Indicates if the QA plug-in accepts ANSI strings. This value is currently ignored and assumed to be TRUE.

Comments

When ApSIC Xbench feeds segments to the QA plug-in, their source and target text may be encoded either in ANSI or Unicode. The *AllowsAnsi* and *AllowsUnicode* fields tell ApSIC Xbench the encodings the QA plug-in is able to handle in case there is more than one possibility.

Please note that saying that a QA plug-in ALLOWS a specific encoding does not mean that it WILL get the text using that encoding. For instance, setting *AllowsUnicode* to TRUE just tells ApSIC Xbench that the encoding is accepted, but it is not being forced to use it. Actually the engine in ApSIC Xbench 2.9 will only send ANSI strings to the QA plug-in.

TQAPluginResult

Contains information about an error detected by the QA plug-in during the QA.

Syntax

```
typedef struct _TQAPluginResult {
   PWSTR Text;
   DWORD SegId;
   PQAPluginResultOptions Options;
 } TQAPluginResult, *PQAPluginResult;
```

Properties

Text: Short description of the error detected.

SegId: ID for the segment having this error.

Options: Pointer to a ***TQAPluginResultOptions*** structure containing extra information about the result. Can be NULL.

Comments

The *Options* element is ignored if this structure is returned from ProcessSegment. It is only checked when returning from GetFirstResult/GetNextResult.

The segment identifier (*SegId*) is provided by ApSIC Xbench through ProcessSegment function calls.

TQAPluginResultOptions

Contains extra information about the result for a QA error.

Syntax

```
  typedef struct _TQAPluginResultOptions {
    WORD StructVersion;
    BOOL Sorted;
    BOOL Groupable;
  } TQAPluginResultOptions,
*PQAPluginResultOptions;
```

Properties

StructVersion: Version number of the *TQAPluginResultOptions* struct. Leave as 0.

Sorted: If FALSE, ApSIC Xbench will sort the results before showing them. If TRUE it will assume they were already sorted by the QA plug-in and their order will not be changed.

Groupable: If FALSE, ApSIC Xbench will not try to pack together repeated errors coming from the QA plug-in, thus showing they all as independent errors.

Comments

TQASegmentInfo

Contains information about the Segment being passed to a function QA plug-in.

Syntax

```
typedef struct _TQASegmentInfo {
  PSTR Source;
  PSTR Target;
  PWSTR WSource;
  PWSTR WTarget;
  DWORD SegId;
} TQASegmentInfo, *PQASegmentInfo;
```

Properties

Source: String with the ANSI version of the source for a segment.

Target: String with the ANSI version of the target for a segment.

WSource: String with the Unicode version of the source for a segment.

WTarget: String with the Unicode version of the target for a segment.

SegId: Identifier for this segment to be used by GetFirstResult/GetNextResult.

Comments

This structure will never have both the ANSI and Unicode versions of a string, only one of the Source/Target pairs will be present (be it the ANSI or the Unicode one) and the other will be NULL. It is up to the QA plug-in to check if the string is valid or NULL. Please note that ApSIC Xbench 2.9 works internally in ANSI mode even when the original files are Unicode, but this changed in ApSIC Xbench 3.0, so do not assume you will always get ANSI strings here.