Sims scripts

Many scripts have been written to operate on Sims content, mostly located under $/tdscontent/scripts. Users should have this directory in their path for all scripts to function correctly. The scripts generally rely on several environment variables, set to point at various directories.

Environment variables

Specific step-by-step instructions for setting up the Korn Shell and the environment variables required for cb scripts are found in sourcesafe under $/simsbuild/code/docs/kornsetup/Readme.doc.

SCENDIR

This variable should be set to your local copy of $/simsbuild/runtime/gamedata. (GameData used to be named TDSScen, hence the SCENDIR). SCENDIR is usually defined in the local Korn Shell variables, in the user profile.ksh.

SPRITEDIR

This directory should be set to your local copy of $/tdscontent/sprites. SPRITEDIR is usually defined in the local Korn Shell variables, in the user profile.ksh.

ACCESSDIR

For running data base imports, this must be set to some local directory. It must be defined in the system variables (autoexec.bat) so that the OMK report can find it from Microsoft Access. Typically, it is set to "C:/TempOMK".

CONTDIR

This variable should be set to your local copy of $/tdscontent. It is used by some scripts to find files in the scripts directory, mainly because perl cannot be invoked automatically on perl script files in Korn Shell 5.

Cb* scripts

Cb stands for "content build". Cb scripts are a collection of short Korn shell programs to be used by artists and other content generators. The scripts serve to easily generate the calls to TDSB and Resource necessary to create and package most content for The Sims. Most of the scripts rely on several environment variables to be setup.

Many cb scripts have an object file as their only argument. The file is the root name of the object, such as "Beds", which includes "Beds.iff" and "Beds.spf". The cb scipts resolve the given filename to an absolute path using SCENDIR, and then perform their functions on the file.

Some cb scripts call TDSB with an OMK file. In this case, they assume that each object file has a directory under SPRITEDIR of the same name that contains an OMK of the same name. For example "cbspr lamps" would call tdsb with "$SCENDIR/objects/lamps –graphics $SPRITEDIR/lamps –spec $SPRITEDIR/lamps/lamps.omk".

cbspr

cbspr (object file)

Generates graphics for the object. Finds the correct OMK file and calls TDSB –graphics.

cbclean

cbclean (object file)

Removes all graphical sprite resources from the object.

cbcleanspr

cbcleanspr (object file)

Shortcut to call cbclean and cbspr in sequence.

cbbmps

cbbmps (object file)

Generates the custom BMP_ resources for an object. Finds the OMK and calls TDSB –bmps.

cbslots

cbslots (object file)

Generates the SLOT resources for an object. Finds the OMK and calls TDSB –slots.

cbimport

cbimport

Finds all exported OMK files in the user's ACCESSDIR and generates the calls to TDSB to process them.

Files ending in:	will call TDSB with the option:
_catalog.omk	-catalog
_sounds.omk	-sounds
_animations.omk	-skills
_ttab.omk	-ttab

The script logs everything to two files. The output, including errors, is $HOME/cbimport.log. Errors only are output to $HOME/cbimporterrors.log. After all OMK files have been processed, the output file is written to stdout, and the error file to stderr. Both files are then removed. The script returns 0 if 1 or more files were imported, and no errors were encountered. It returns 1 if no files were found to import. It returns 2 if any file had an error on import.

cbimporterr

cbimporterr

Wraps cbimport and reports the end status and errors, if any. Also, the errors are run through the "more" utility, which shows the text one screen at a time. This allows Windows98 users to actually see the results. Afterwards, the log files $HOME/cbimp.log and $HOME/cbimp_errs.log are not removed.

cbsprall

cbsprall

Runs cbcleanspr on all objects in the game and reports the differences between the resulting object file and the object in sourcesafe. Cbsprall operates in the current directory and outputs a text file for each object SPF file in the game. The text file contains all the results of getting the relevant files from sourcesafe, running cbcleanspr, and running "resource diff" on the two files. As the script progresses, the name of each object file is written to stdout.

It was intended that cbsprall would be run automatically every night in order to verify that the game's sprite content could be fully regenerated with no differences. However, this goal was never achieved. Firstly, TDSB has several asserts from within the debug memory manager that will stop the automated run after the first couple of objects. Secondly, after ignoring the asserts, there are many differences that we did not have time to deal with.

Cbsprall relies on the utility scripts ss_file_match.ksh and ss_file_match.pl, found in the same directory as the cb scripts. These basically do pattern matching on the output of a recursive sourcesafe directory listing (ss dir –r), making it simple to find, for example, all files ending in SPF under GameData/Objects.

cbprep

Called internally by some cb scripts to make sure common environment variables are set up.

Unused cb scripts

Many cb scripts are no longer used. Some are obviated by data base exports, and others perform functions that are no longer needed.

cbattr – served only to embed attribute labels from an OMK. Now done in edith.

cbcareers – served to create career data in careers.iff from an OMK. Careers were changed to be read in from a text file, so this is no longer needed.

cbcatalog – replaced by export/import system with cbimport.

cbcopy – spawns an object file. TDSB is now used directly for this.

cbcopybh – copies behavior resources from one object file to another. Not practical since other resources depend on the behaviors and vice versa.

cbcopyperson – makes a copy of a person. TDSB is now used directly for this, like cbcopy.
cbfloors – replaced by different floor scripts.
cbmakedir – old script to create a new sprite directory for an object. Not needed since directories are set up by a programmer.
cbmultispr – attempt at something old. Stubbed and not used.
cbnew – creates a copy of object "vanilla". This no longer exists.
cbpeople – builds attribute labels for all people. Now done in edith.
cbskills – replaced by export/import system with cbimport.
cbsounds – replaced by export/import system with cbimport.
cbstub – searches through OMK files for TGA file usage and copies a stub TGA file to the object's sprite directory. Not practical since stub TGA's are so large.
cbterrain – compresses terrain sprites into terrain.iff. Terrain no longer uses sprites.
cbwalls – replaced by different wall scripts.

check_btrees.pl

perl check_btrees.pl(behavior text file) ...

Operates on stdin, which should be one of the TXT files output by edith's "Save Behaviors As Text" function. It checks for semantic problems with behavior trees. Currently it only checks for animate primitives that specify 0 expected events yet define some event handling.

check_skill

check_skill (skill name segment)

Finds all CMX files that refer to the given string, and all CFP files whose name contains the string,

cleancharacters

cleancharacters

Resets all generated face shots for people. The script finds all IFF files in "$SCENDIR/objects/people" and "$SCENDIR/../userdata/characters" and removes all BMP_ resources from them.

convert_resfile

convert_resfile (resfile root name)

Makes a fresh copy of the given resource file using "resource copy". The benefit of using this script is that it hides the ResolveResFile during the copy and then restores it. Since ResolveResFile is read only, this prevents the resolve resources from getting merged into the new IFF file. The only reason one would want to make a fresh copy of a resource file is for version upgrades. For example, when IFFResFile went from 1.0 to 2.0, this script was run on all data files.

convert_all_resfiles

convert_all_resfiles

Calls convert_resfile for all IFF files in the current directory and subdirectories.

diff_directory

diff_directory (directory1) (directory2) (command) [(glob pattern) ...]

Compares the contents of the two given directories using the given command. If one or more glob patterns are specified, they are passed directly to the "find" command to determine which files are to be compared. (Don't forget to escape special characters.) Otherwise, all files in the directory and subdirectories are compared.

res_diffdir

res_diffdir(directory1) (directory2)

Calls diff_directory on the given directories with "resource diff" as the command.

sims_diff_report

sims_diff_report [(days back)]

Specific to the Sims sourcesafe structure, this uses various sourcesafe commands to get older versions of some sensitive files and a history report of others. It operates on the current directory and generates a various report files. A full description of the files may be found in the separate document SimsDiffReport.doc.

dump_bmp

dump_bmp (resource file) ...

Dumps out all BMP_ resources in the given file list. Resources are dumped to the file name appended by the resource id appended by ".BMP".

res_diff_current

res_diff_current (IFF file) ...

Reports differences for each specified IFF (or SPF) file to the corresponding file in sourcesafe. The script creates a temporary directory in $TEMP. NOTE: the sourcesafe working directory must be set to the same directory where the given file resides, therefore the command can only be used on one sourcesafe directory at a time. The script is for programmers to be able to see what resources changed before checking in a resource file. Typical usage would be something like this:

```
ss cd $/simsbuild/runtime/gamedata/objects
cd $HOME/simsbuild/runtime/gamedata/objects
res_diff_current lamps.iff beds.iff
```

ss_file_match

ss_file_match (ss dir) (cache dir)

ss_file_match (ss dir) (cache dir) [(regular expression)]

Maintains a cache of recursive sourcesafe directory listings in the given cache directory of all the files in the given ss directory. If a regular expression is specified, the script calls ss_file_match.pl on the cached listing to report matches. Otherwise, the cache is removed.

For example, I use this to find source files in sourcesafe without having to remember which directory the files are in:

```
alias xf='ss_file_match $/simsbuild $HOME/ss_cache'
xf ^person.cpp ^object.cpp
```

This outputs:

```
$/SimsBuild/Code/msrc/Objects/object.cpp
$/SimsBuild/Code/msrc/Objects/person.cpp
```

ss_file_match.pl

perl ss_file_match.pl (directory listing) [(regular expression) ...]

Searches the given directory listing file (output by "ss dir –r") and matches the items against the given list of regular expressions. If a slash occurs in the regular expressions, then the full path of each item is compared. Otherwise, only the filename is compared, ignoring the directory name.

zzz_test

zzz_test (directory)

Calls zzz_test.pl for every TXT file found in the given directory. The TXT files are parsed as Slang output files. Before calling zzz_test.pl, the file is backed up to the same location with ".BAK" appended. Upon importing the TXT files back into the game, it is easy to spot "untranslated" text, that is, text that does not have ZZZ's in it.

zzz_test.pl

Processes stdin as a slang output file, replaces all occurrences of certain strings with a ZZZ-modified string so that as many different types of string schemes are given ZZZ text as possible. The strings considered are those inside slang bracketing characters, currently ">>>>" and "<<<<". The 3 replacements are as follows:

- Each slash character is replaced with "/ZZZ".
- Each percent character is replaced with "ZZZ%".
- Each string, X, is replaced with "ZZZXZZZ"

Unused scripts

A few scripts are not in common usage, but were left around for possible scavenging or later upgrading.

enumresources

enumresources (resource file) (type)

For the given resource file and resource type, prints to stdout a list of all the id numbers of resources of that type in that file. Relies on the output of parse_resinfo.pl.

extract_tags.pl

perl extract_tags.pl (tag) ...

Processes stdin as an OMK and reports all instances of the specified tags.

parse_resinfo.pl

perl parse_resinfo.pl

Processes stdin as the output of "resource info" command and prints out the type and id of each reported resource on a separate line.

parse16

parse16 (file) (type) (id) [(label file) (label resource id)]

Parses the resource specified by file, type and id as an array of 16 bit integers and prints the result on stdout. If a label file and label resource id are specified, then the STR# resource of that id in that file are used as labels for the elements of the array. Otherwise, numbers are used. Relies on the output of "resource binprint" and uses the perl script "parse16.pl".

parse16.pl

Used by parse16.ksh, it takes a file containing the output of "resource binprint" and a label file and outputs a labeled array of 16 bit hex values.

print_objd

print_objd (file) (id)

Uses parse16 and the Sims behavior labels files ($SCENDIR/behavior) to print out the specified OBJD resource as text.

printall_objd

printall_objd (file) ...

Uses enumresources and print_objd to print out all the OBJD resources in the given files as text.

search_objd

search_objd (expression)

Calls printall_objd and uses a short perl program to print lines with an occurrences of the given expression.

showfuncs.pl

Old script to output all functional tree entry points. Entry points were moved out of OBJD resources, so this script no longer works.

stubtga

Old script to generate a stub TGA file for each reference to a TGA file from an OMK file. Not practical since stub TGA's are so large.

enumresources.pl

Stray script. Copy of parse_resinfo.

fill_strings

Old script that did something with behavior strings.

glslots

Old script to make global slots. Now "cbslots global" does it.

glsounds

Old script to make global sounds. Now cbimport does it.

glspr

Old script to make global sprites. Now "cbcleanspr global" does it.

makeicons

Really old script to make icons for old Sims characters.

parse_omk_report.pl

Old script to parse the giant OMK report that used to be output by the object DB.

ss_add_object

Old script to add all content and runtime files for a given object.