Skeleton Parser 2008 (Beta V0.0.1)

Introduction

The ISPF Skeleton Parser reads file tailoring input files (skeletons) and generates several cross reference files:

- Skeleton file-to-variable cross reference
- Skeleton file-to-embedded skeleton file cross reference
- Skeleton file-to-ISPF skeleton built-in function cross reference

For each cross reference dataset, three optional file formats can be created:

- XML records
- Comma separated value records, suitable for loading into an Excel spreadsheet
- Fixed format records, suitable for loading into relational tables.

This version of the parser is not an update to any previous version. At present there is neither a graphic user interface front end nor any query capabilities. The parser itself runs as a command-prompt .exe file.

Installation

There are two installation files:

- SETUP.EXE
- SkeletonUtilitySetup.MSI

You must have administrator rights to the system on which you will be installing the parser. Also, you must have the right to install applications.

To install the parser, simply run the setup program. Note that this program may install the Microsoft .NET runtime. This runtime will be downloaded from Microsoft, therefore the installer needs Internet access. It is the installation of the .NET runtime that requires administrator rights on your PC.

Following the runtime installation the SETUP.EXE program will install the parser and its supporting files. You will be able to override the install destination directory. The rest of the installation is automated. All of the installation files related to the utility (save for the .NET runtime) are installed in a single directory.

The application directory installed files are:

Base.xml	A sample XML file. You can use this as a base copy. If you accidentally wipe out the base copy there is a backup in the Samples directory.
imbeds2.ico	The icon for the uninstaller
Library.dll	Program support library

LibraryConcatenation.dll	Program support library.
Parser.dll	Program support library
README.PDF	Updated program information (this file).
SkeletonParserDS.dll	Program support library
SkeletonParser.exe	The parser program.
SkeletonParserQuery.exe	A GUI query program for displaying the Skeleton-Variable and Skeleton-Skeleton cross-references.
SkelParser.cmd	A sample CMD file to invoke the parser program.

Below the application directory are two other directories, Samples and Doc.
This documentation is in the Doc directory.

The Samples directory installed files are:

BASE.xml	A read only sample XML parameters file. This should not be modified, but can be copied to create new configurations.
BASE.xsd	A read only schema for the XML file
parser.rex	A sample Regina REXX program showing how to parse the program output.
parserDB2.rex	A sample Regina REXX program showing how to parse the fields offsets fixed format file to create DB2 *CREATE TABLE* control cards.
SkelParserDB2R.cmd	Invokes parserDB2.rex, passing in a configuration XML file name.
SkelParserR.cmd	Invokes parser.rex, passing in a configuration XML file name.

Customization

You must create a separate directory on the target PC system for each host PDS. There must be one dataset for each member of the PDS. At this time, no utilities are supplied to facilitate transferring files from the z/OS host to the PC.

The parser supports multiple host PDS concatenations. Each unique concatenation of datasets must be specified in a separate XML file. A sample XML file has been provided for customization. This is Config1.XML. Config1.xml is a copy of BASE.XML. Note that Config1.XML and BASE.XML are identical, but BASE.XML is marked READONLY. You should leave BASE.XML as a model for any configurations you want to create. But if you copy BASE.XML, be sure to change the file properties of the newly created file to remove the READONLY property.

XML Tags

<QueryDisplayName>

A character string that is used in the Query GUI to identify the configuration on all of the dialog windows. Note that this does *not* have to match the <ConfigurationName> value described below in the fixed file output tags definitions.

<Directory>

Find the <Directory> tag. In BASE.XML it is:

```
<Directory>
    <DirectoryName>C:\mypath\mycustomSkels</DirectoryName>
    <DirectoryLabel>Custom</DirectoryLabel>
    <DirectoryHostName>somnode.CUSTOM.SKELS</DirectoryHostName>
  </Directory>
  <Directory>
    <DirectoryName>C:\mypath\myvendorSkels</DirectoryName>
    <DirectoryLabel>Vendor</DirectoryLabel>
    <DirectoryHostName>somnode.VENDOR.SKELS</DirectoryHostName>
  </Directory>
```

You may specify one or more directories. Each represents one host ISPF skeleton PDS. The order of the <Directory> tags should match the order of the concatenation that is being simulated. The above XML fragment represents this JCL:

```
//ISPSLIB DD
DISP=SHR,DSN=somnode.CUSTOM.SKELS
```

<DirectoryName>

This is the PC directory containing the datasets copied from one host PDS. At the time the parser is run, this directory must exist, or the parser will terminate with an error. The directory does not have to be on the C: drive, and it can be on a network drive that is read accessible. Note that reading files in network folders from a parser running on a remote machines will be a slower process.

<DirectoryLabel>

The directory labels are not critical to the program operation. They are written as specified to one of the output files. The label is just a way to give a more meaningful title to the specified Host/PC file combination in the <Directory> element.

<DirectoryHostName>

Like the <DirectoryLabel>, the data supplied for the <DirectoryHostName> is not critical to the program operation. It is useful for identifying what host file was used to fill the members in the PC directory.

Output file types

There are three output file types: CSV, XML and fixed format. Within each of the file types there are several output datasets.

Variables

The Variables dataset contains fields showing the Skeleton-Variable relationships.

Skeletons

The Skeletons dataset contains fields showing the Skeleton-)IM'd skeleton relationships.

Functions

Each function is shown in the Variables reference. But the Functions dataset contains extra information, showing the arguments passed to each function.

Libraries

The information in the Libraries file is directly extracted from the XML <Directory> elements in your input.

Keywords

For the fixed format file output, the Keywords file contains information that can be used to build a look-up table. Most keywords are ISPF Skeleton commands, such as IM, SEL, SET or DO. But there are some that further delineate some of the commands. Thus DOWHILE and DOUNTIL are refinements to how a)DO statement can be written, but neither DOWHILE nor DOUNTIL are ISPF commands.

The non-ISPF keywords are: DOWHILE, DOUNTIL and DATALINE.

VarTypes

The VarTypes dataset is a lookup table for variable type numbers used in the Variables dataset. This dataset is only generated if the fixed format output files are generated.

VARIABLE_REFERENCE

Variable found in a non-command line.

CONSTANT_REFERENCE

A constant found in a)SET or)SETF statement. Up to the first eight characters are shown.

CONSTANT_TEST

A constant found in a)SEL or)IF statement. Up to the first eight characters are shown.

VARIABLE_TEST

A variable used in a)SEL or)IF statement

LVAL_ASSIGNMENT

The left side (LVAL) of an assignment statement,)SET or)SETF. The variable is specified without a leading &.

LVAL_INDIRECT_ASSIGNMENT

The left side (LVAL) of an assignment statement,)SET or)SETF. The variable name has a leading &, so it is an indirect assignment. For example:

```
)SET FRED = ETHEL
)CM The following is the same as )SET ETHEL = 3
)SET &FRED = 3
```

RVAL_CONSTANT_ASSIGNMENT

The right side (RVAL) of an assignment statement is a constant.

RVAL_VARIABLE_ASSIGNMENT

The right side (RVAL) of an assignment statement is a variable.

LVAL_NULL_ASSIGNMENT

The)SET or)SETF statement is a null (&Z) assignment. The LVAL variable has no leading &.

LVAL_INDIRECT_NULL_ASSIGNMENT

The)SET or)SETF statement is a null (&Z) assignment. The LVAL is indirect (variable name has a leading &).

REXX_CALL_CONSTANT_REFERENCE

A constant used in an in stream)REXX statement.

REXX_CALL_VARIABLE_REFERENCE

A variable used in an in stream)REXX statement.

REXX_PROCEDURE

An external procedure specified in a)REXX statement

FUNCTION

The listed variable is an ISPF skeleton built-in function.

FieldOffsets

The FieldOffsets file is only generated for fixed format output. It contains the field names, types and offsets of the data in each file. This file can be used to generate load datasets for relational databases.

```
Table: Variables
....+....0....+....0....+....0....+....0....+....0....+....0....+....0....+....
0
Configuration                  001 030 030 String          True
ConfigNum                      031 003 002 Int16           True
Variable                       034 008 008 String          True
Skeleton                       042 008 008 String          True
LineNumber                     050 005 004 Int32           True
Position                       055 002 002 Int16           True
Command                        057 009 009 String          False
CommandCode                    066 003 002 Int16           False
Type                           069 025 025 String          False
TypeCode                       094 003 002 Int16           False
Table: Skeletons
....+....0....+....0....+....0....+....0....+....0....+....0....+....0....+....
0
Configuration                  001 030 030 String          True
ConfigNum                      031 003 002 Int16           True
Skeleton                       034 008 008 String          True
SkelOffset                     042 002 002 Int16           False
ChildSkeleton                  044 025 025 String          True
ChildSkelOffset                069 002 002 Int16           False
LineNumber                     071 005 004 Int32           True
OPT                            076 001 001 Boolean         False
NOFT                           077 001 001 Boolean         False
EXT                            078 001 001 Boolean         False
Table: Functions
....+....0....+....0....+....0....+....0....+....0....+....0....+....0....+....
0
Configuration                  001 030 030 String          True
ConfigNum                      031 003 002 Int16           True
Function                       034 008 008 String          True
Skeleton                       042 008 008 String          True
LineNumber                     050 005 004 Int32           True
Pos                            055 002 002 Int16           True
Argument                       057 060 060 String          False
Table: Libraries
....+....0....+....0....+....0....+....0....+....0....+....0....+....0....+....
0
Configuration                  001 030 030 String          True
ConfigNum                      031 003 002 Int16           True
Offset                         034 002 002 Int16           True
UserTag                        036 050 050 String          False
HostFileName                   086 044 044 String          False
PCFileName                     130 256 256 String          False
Table: Keywords
....+....0....+....0....+....0....+....0....+....0....+....0....+....0....+....
0
Code                           001 003 002 Int16           True
```

The table names are only suggestions. They match the generated low level qualifier of the above specified output file names. The fields in the file are defined thus:

\

Column	Length / Type	Description
1	32 / character	Field Name
34	3 / numeric	Offset (first field is always at offset 1)
38	3 / numeric	Display length
42	3 / numeric	Internal length
46	15 / Character	Type Int16 = 2 byte integer Int32 = 4 byte integer String = variable length character string.

Column	Length / Type	Description
62	5	Key field indicator (True or False)

<ExcelOutput>

```
<ExcelOutput>

<ExcelOutputFolder>C:\myCSVOutputPath\</ExcelOutputFolder>
   <ExcelOutputHLQ>cProd</ExcelOutputHLQ>
```

Use the <ExcelOutput> element to define the characteristics of a set of output files consisting of comma separated values (CSV). These files can be loaded into a spreadsheet program, such as MS Excel. If this element is not defined, then no CSV files will be generated or written.

<ExcelOutputFolder>

Use this element to define the full path to the directory in which the files will be generated. At the time the parser is run, this directory must exist, or the parser will terminate with an error.

<ExcelOutputHLQ>

Use this element to define the High Level Qualifier(HLQ) of the CSV output datasets. To this HLQ will be appended one of the output dataset types: Variables, Skeletons, Functions or Libraries. The Keywords, VarTypes and FieldOffset files are not generated for CSV output. For the output folder and HLQ shown above, the following files would be generated in the directory C:\myCSVOutputPath

cProd.Variables.csv
cProd.Skeletons.csv
cProd.Functions.csv
cProd.Libraries.csv

<XMLOutput>

```
<XMLOutput>

<XMLOutputFolder>C:\myXMLOutputPath</XMLOutputFolder>
    <XMLOutputHLQ>xProd</XMLOutputHLQ>
  </XMLOutput>
```

Use the <XMLOutput> element to define the characteristics of a set of output files consisting of XML tags for the parsed data. Some databases can be defined with XML file input. Newer languages can directly query data in XML files.

<XMLOutputFolder>

Use this element to define the full path to the directory in which the files will be generated. At the time the parser is run, this directory must exist, or the parser will terminate with an error.

<XMLOutputHLQ>

Use this element to define the High Level Qualifier(HLQ) of the XML output datasets. To this HLQ will be appended one of the output dataset types: Variables, Skeletons, Functions or Libraries. The Keywords, VarTypes and FieldOffset files are not generated for XML output. For the output folder and HLQ shown above, the following files would be generated in the directory C:\myCSVOutputPath

xProd.Variables.xml
xProd.Skeletons.xml
xProd.Functions.xml
xProd.Libraries.xml

<FixedOutput>

```
  <FixedOutput>
    <Configuration>
      <ConfigurationName>MyConfigName</ConfigurationName>
      <ConfigurationNumber>1</ConfigurationNumber>
    </Configuration>


<FixedOutputFolder>C:\myFixedFilesOutputPath</FixedOutputFolder>
    <FixedOutputHLQ>fProd</FixedOutputHLQ>
```

Use this compound element to define the characteristics of a set of output files that can be used to load tables for a relational database such as DB2 or MS SQL Server.

<ConfigurationName> <ConfigurationNumber>

The reason for generating the fixed file input is create data to load relational database tables. The configuration name and number exist to allow output from different runs of the parser to be stored in the same tables. Use the Configuration data to retrieve the proper data for the concatenations you defined.

This might be clearer with an example.

If you have a test system with a test skeletons library and then a production skeletons concatenated following it, you can define a test configuration thus:

```
  <Directory>
    <DirectoryName>C:\mypath\test</DirectoryName>
    <DirectoryLabel>Custom</DirectoryLabel>
    <DirectoryHostName>P.CUSTOM.SKELS</DirectoryHostName>
  </Directory>
  <Directory>
    <DirectoryName>C:\mypath\vendor</DirectoryName>
    <DirectoryLabel>Vendor</DirectoryLabel>
    <DirectoryHostName>P.VENDOR.SKELS</DirectoryHostName>
  </Directory>
      ⋮
  <FixedOutput>
    <Configuration>
      <ConfigurationName>Test</ConfigurationName>
      <ConfigurationNumber>2</ConfigurationNumber>
    </Configuration>


<FixedOutputFolder>C:\myFixedFilesOutputPath</FixedOutputFolder>
    <FixedOutputHLQ>test</FixedOutputHLQ>
  </FixedOutput>
```

With a production system that only uses the production library, the XML for the configuration would look thus:

```
  <Directory>
    <DirectoryName>C:\mypath\vendor</DirectoryName>
    <DirectoryLabel>Vendor</DirectoryLabel>
    <DirectoryHostName>P.VENDOR.SKELS</DirectoryHostName>
  </Directory>
      ⋮
  <FixedOutput>
    <Configuration>
      <ConfigurationName>Production</ConfigurationName>
      <ConfigurationNumber>1</ConfigurationNumber>
    </Configuration>


<FixedOutputFolder>C:\myFixedFilesOutputPath</FixedOutputFolder>
    <FixedOutputHLQ>prod</FixedOutputHLQ>
  </FixedOutput>
```

Note that the test and production configurations share an output folder, but that the HLQs of *test* and *prod* keep the output files distinct.

<FixedOutputFolder>

Use this element to define the full path to the directory in which the files will be generated. At the time the parser is run, this directory must exist, or the parser will terminate with an error.

<FixedOutputHLQ>

Use this element to define the High Level Qualifier(HLQ) of the fixed output datasets. To this HLQ will be appended one of the output dataset types: Variables, Skeletons, Functions, Libraries, Keywords, VarTypes and FieldOffsets. For the output folder and HLQ shown above, the following files would be generated in the directory C:\myFixedFilesOutputPath

fProd.Variables.txt
fProd.Skeletons.txt
fProd.Functions.txt
fProd.Libraries.txt
fProd.Keywords.txt
fProd.VarTypes.txt
fProd.FileOffsets.txt

<IgnoreSkel>

You may specify zero, one or more skeleton names that are to be ignored in the parsing. Use one <IgnoreSkel> element per skeleton. Only specify the eight character skeleton name, not the extension. The utility ignores file extensions.

<DebugInputLine>

This is a true/false element. Setting the value to *true* will cause the input skeleton line to be written to the CSV and XML files. This element has no effect on the fixed format file output. Note that this is not needed, but may be requested as documentation if you find that one or more skeleton lines are not being parsed correctly.

Creating Different Runtime Versions

Each configuration needs a separate run time version. Since all of the information necessary to create a set of output files for a version is in an XML file customized as outlined above, the only run time parameter the parser needs is the name of the XML file it is to process.

Sample batch files have been supplied showing how the utility is invoked. The Samples directory also contains files written in REXX showing how to invoke the parser from REXX. You must have a REXX interpreter installed on your system to use these files. The samples were tested with Regina REXX, which is a free, open source implementation of a REXX interpreter.