# ISPF File Tailoring Skeleton Parser

[ISPF File Tailoring Skeleton Parser 1](#_Toc268536715)

[Introduction 3](#_Toc268536716)

[Installation 3](#_Toc268536717)

[The Parser Engine 4](#_Toc268536718)

[Customization 4](#_Toc268536719)

[XML Tags 5](#_Toc268536720)

[<QueryDisplayName> 5](#_Toc268536721)

[<Directory> 5](#_Toc268536722)

[<DirectoryName> 5](#_Toc268536723)

[<DirectoryLabel> 5](#_Toc268536724)

[<DirectoryHostName> 5](#_Toc268536725)

[Output file types 5](#_Toc268536726)

[Variables 6](#_Toc268536727)

[Skeletons 6](#_Toc268536728)

[Tables(DOT) 6](#_Toc268536729)

[Functions 6](#_Toc268536730)

[Libraries 6](#_Toc268536731)

[Keywords 6](#_Toc268536732)

[VarTypes 6](#_Toc268536733)

[FieldOffsets 8](#_Toc268536734)

[<ExcelOutput> 9](#_Toc268536735)

[<ExcelOutputFolder> 9](#_Toc268536736)

[<ExcelOutputHLQ> 9](#_Toc268536737)

[<XMLOutput> 10](#_Toc268536738)

[<XMLOutputFolder> 10](#_Toc268536739)

[<XMLOutputHLQ> 10](#_Toc268536740)

[<FixedOutput> 11](#_Toc268536741)

[<ConfigurationName> <ConfigurationNumber> 11](#_Toc268536742)

[<FixedOutputFolder> 12](#_Toc268536743)

[<FixedOutputHLQ> 12](#_Toc268536744)

[<IgnoreSkel> 12](#_Toc268536745)

[<Unimbedded Skeleton> 13](#_Toc268536746)

[<ImbedRecurseFail> 13](#_Toc268536747)

[<DebugInputLine> 14](#_Toc268536748)

[Creating Different Run Time Versions 14](#_Toc268536749)

[The Graphical Front-End 16](#_Toc268536750)

[Load a New Configuration 17](#_Toc268536751)

[Skeleton to Variable Cross-Reference 18](#_Toc268536752)

[Displaying the Skeleton 19](#_Toc268536753)

[Exporting data 20](#_Toc268536754)

[Skeleton to Skeleton Cross-Reference 21](#_Toc268536755)

[Display the Skeleton 22](#_Toc268536756)

[Exporting data 23](#_Toc268536757)

[Skeleton to Table (DOT) cross-reference 24](#_Toc268536758)

[Display the Skeleton 25](#_Toc268536759)

[Exporting data 25](#_Toc268536760)

[Skeleton to Program Cross-Reference 27](#_Toc268536761)

[Display the Skeleton 28](#_Toc268536762)

[Exporting data 28](#_Toc268536763)

[Skeleton Expansion 30](#_Toc268536764)

[Expansion Context Menu 31](#_Toc268536765)

[Filtering Text Within a skeleton line 31](#_Toc268536766)

[Using a Filter to Hide Skeleton Expansions 32](#_Toc268536767)

[Removing Filters 33](#_Toc268536768)

[Positioning on a Specific Line (Goto) 33](#_Toc268536769)

[Skeleton Logic Display For A Selected Line 35](#_Toc268536770)

[Variables Display Within Lines 35](#_Toc268536771)

[Graphic View 36](#_Toc268536772)

## Introduction

The ISPF File Tailoring 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 a limited graphic user interface front-end. The parser itself runs as a command-prompt .exe file and it is invoked internally by the graphic front-end.

## Installation

There are two installation files:

* SETUP.EXE
* INSTALLER.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 will install the Microsoft .NET 3.5 run time. This run time will be downloaded from Microsoft, therefore the installer needs Internet access. It is the installation of the .NET run time that requires administrator rights on your PC.

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

The application directory installed files are:

Config1.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.TXT Updated program information.

SkeletonParserDS.dll Program support library

SkeletonParser.exe The parser program.

SkeletonParserQuery.exe The Graphical Front End for the parser engine (see below). This is a GUI query program for displaying the Skeleton-Variable cross reference, Skeleton-Skeleton cross-reference and the Skeleton Expansion as table data.

SkelParser.cmd A sample CMD file to invoke the parser program.

Utility.dll Program support library

XMLParmsReader.dll Program support library

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

SkeletonParser\_Readme.PDF This documentation.

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.

## The Parser Engine

The parser.dll file implements the parser engine. The engine does the actual analysis of skeleton data passed in the XML parameter file. The parser engine generates internal data stores that are then used to create the various output or query data defined below. The parser engine can be invoked by the batch parser and the Graphical Front End (GFE).

## 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 Graphical Front End 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

// DD DISP=SHR,DSN=somnode.VENDOR.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 machine 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 (comma separated values), 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.

#### Tables(DOT)

The DOT dataset contains fields showing the Skeleton-)DOT (table) 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 False

ConfigNum 031 005 002 Int16 False

Variable 036 072 072 String False

Skeleton 108 008 008 String False

LineNumber 116 006 004 Int32 False

Position 122 005 002 Int16 False

Command 127 009 009 String False

CommandCode 136 005 002 Int16 False

Type 141 030 030 String False

TypeCode 171 005 002 Int16 False

Table: Skeletons

....+....0....+....0....+....0....+....0....+....0....+....0....+....0....+....0

Configuration 001 030 030 String False

ConfigNum 031 005 002 Int16 False

Skeleton 036 008 008 String False

SkelOffset 044 005 002 Int16 False

ChildSkeleton 049 025 025 String False

ChildSkelOffset 074 005 002 Int16 False

LineNumber 079 006 004 Int32 False

OPT 085 001 001 Boolean False

NOFT 086 001 001 Boolean False

EXT 087 001 001 Boolean False

Table: Functions

....+....0....+....0....+....0....+....0....+....0....+....0....+....0....+....0

Configuration 001 030 030 String False

ConfigNum 031 005 002 Int16 False

Function 036 008 008 String False

Skeleton 044 008 008 String False

LineNumber 052 006 004 Int32 False

Position 058 005 002 Int16 False

Argument 063 060 060 String False

Table: Libraries

....+....0....+....0....+....0....+....0....+....0....+....0....+....0....+....0

Configuration 001 030 030 String False

ConfigNum 031 005 002 Int16 False

Offset 036 005 002 Int16 False

UserTag 041 050 050 String False

HostFileName 091 044 044 String False

PCFileName 135 256 256 String False

Table: Keywords

....+....0....+....0....+....0....+....0....+....0....+....0....+....0....+....0

CommandCode 001 005 002 Int16 True

Command 006 020 020 String False

Table: VarTypes

....+....0....+....0....+....0....+....0....+....0....+....0....+....0....+....0

TypeCode 001 005 002 Int16 True

Type 006 030 030 String False

Table: Configurations

....+....0....+....0....+....0....+....0....+....0....+....0....+....0....+....0

ConfigNum 001 005 002 Int16 True

Configuration 006 030 030 String False

Table: DOT

....+....0....+....0....+....0....+....0....+....0....+....0....+....0....+....0

Configuration 001 030 030 String False

ConfigNum 031 005 002 Int16 False

Skeleton 036 008 008 String False

TableName 044 030 030 String False

LineNumber 074 006 004 Int32 False

Table: Programs

....+....0....+....0....+....0....+....0....+....0....+....0....+....0....+....0

Configuration 001 030 030 String False

ConfigNum 031 005 002 Int16 False

Skeleton 036 008 008 String False

ProgramName 044 030 030 String False

LineNumber 074 006 004 Int32 False

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. |
| 62 | 5 | Key field indicator (True or False) |

See the sample Rexx file ParserDB2.REX. This Regina Rexx program processes the above offsets file to create z/OS DB2 DDL and utility LOAD statements.

### <ExcelOutput>

<ExcelOutput>

<ExcelOutputFolder>C:\myCSVOutputPath\</ExcelOutputFolder>

<ExcelOutputHLQ>cProd</ExcelOutputHLQ>

</ExcelOutput>

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.

The ExcelOutput related tags are ignored when the XML file is used as the control for the Graphical User Interface.

### <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.

As noted above, the ExcelOutput related tags are ignored when the XML file is used as the control for the Graphical User Interface.

### <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, Tables (DOT), Programs (skeleton to executed program cross reference), 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](file:///C:\myCSVOutputPath)

cProd.Variables.csv

cProd.Skeletons.csv

cProd.DOT.csv

cProd.Programs.csv

cProd.Functions.csv

cProd.Libraries.csv

As noted above, the ExcelOutput related tags are ignored when the XML file is used as the control for the Graphical User Interface.

### <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.

The XMLOutput related tags are ignored when the XML file is used as the control for the Graphical User Interface.

### <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.

As noted above, the XMLOutput related tags are ignored when the XML file is used as the control for the Graphical User Interface.

### <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, Tables (DOT), Programs (skeleton to executed program cross reference), 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](file:///C:\myCSVOutputPath)

xProd.Variables.xml

xProd.Skeletons.xml

xProd.DOT.xml

xProd.Programs.xml

xProd.Functions.xml

xProd.Libraries.xml

As noted above, the XMLOutput related tags are ignored when the XML file is used as the control for the Graphical User Interface.

### <FixedOutput>

<FixedOutput>

<Configuration>

<ConfigurationName>MyConfigName</ConfigurationName>

<ConfigurationNumber>1</ConfigurationNumber>

</Configuration>

<FixedOutputFolder>C:\myFixedFilesOutputPath</FixedOutputFolder>

<FixedOutputHLQ>fProd</FixedOutputHLQ>

</FixedOutput>

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.

Only the ConfigurationName and ConfigurationNumber tags are used when the XML file is used as the control for the Graphical User Interface. The FixedOutput related file and folder tags are ignored when the XML file is used as the control for the Graphical User Interface.

### <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>fTest</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>fProd</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.

As noted above, the FixedOutput related file and folder tags are ignored when the XML file is used as the control for the Graphical User Interface.

### <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, Tables (DOT), Programs (skeleton to executed program cross reference), 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](file:///C:\myCSVOutputPath)

fProd.Variables.txt

fProd.Skeletons.txt

fProd.DOT.txt

fProd.Programs.txt

fProd.Functions.txt

fProd.Libraries.txt

fProd.Keywords.txt

fProd.VarTypes.txt

fProd.FileOffsets.txt

As noted above, the FixedOutput related file and folder tags are ignored when the XML file is used as the control for the Graphical User Interface.

### <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.

### <Unimbedded Skeleton>

You may specify zero, one or more skeletons that are to be added to the Skeleton-to-Skeleton cross reference table. These skeletons are normally shown in the table as being )IM'd by “no parent”. You can change the parent to the name of a process or program, which would otherwise be unknown by the parser.

<Unimbedded Skeleton="CMN$$AUD" Parent="..AUDIT Program"/>

<Unimbedded Skeleton="CMN$$AUD" Parent="..AUDIT Program"/>

<Unimbedded Skeleton="CMNIMRPM" Parent="..PROMOTION"/>

<Unimbedded Skeleton="CMNIMPRM" Parent="..PROMOTION"/>

<Unimbedded Skeleton="CMN$$D2J" Parent="..DB2Jobs" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN$$OTH" Parent="LIBTYPE\_OTHER" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN10" Parent="..Distribution" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN11" Parent="..Distribution" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN14" Parent="..Distribution" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN15" Parent="..Distribution" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN17" Parent="..Distribution" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN18" Parent="..Distribution" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN19" Parent="..Distribution" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN20" Parent="..Install" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN20I" Parent="..Install" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN21" Parent="..Install" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN24" Parent="..Install" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN25" Parent="..Install" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN28" Parent="..Install" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN29" Parent="..Install" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN30" Parent="..Baseline" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN30I" Parent="..Baseline" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN31T" Parent="..Temp" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN31TI" Parent="..Temp" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN32" Parent="..Baseline" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN34T" Parent="..Temp" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN35T" Parent="..Temp" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN37" Parent="..Baseline" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN38T" Parent="..Temp" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN39T" Parent="..Temp" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN49" Parent="..Backout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN50" Parent="..Backout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN50I" Parent="..Backout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN50T" Parent="..TempBkout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN50TI" Parent="..TempBkout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN51" Parent="..Backout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN54" Parent="..Backout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN55" Parent="..Backout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN55I" Parent="..Backout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN55T" Parent="..TempBkout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN55TI" Parent="..TempBkout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN56" Parent="..Backout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN57" Parent="..Backout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN58" Parent="..Backout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN59" Parent="..Backout" NOFT="false" OPT="false" EXT="false"/>

<Unimbedded Skeleton="CMN64" Parent="..Revert" NOFT="false" OPT="false" EXT="false"/>

The Unimbedded element has five attributes: Skeleton, Parent, NOFT, OPT and EXT. The Skeleton is the name of a skeleton that is not )IM'd by any other skeleton in the concatenation. The Parent is the name of a process or dialog that uses the ISPF FTINCL to invoke the skeleton. You may specify up to 40 characters here. If longer strings are specified, they are truncated to 40 characters. NOFT, OPT and EXT are the file tailoring options for “no file tailoring”, “optional” and “extend ON”. These have either a *true* or a *false* value. The default is false.

The Unimbedded element must have at the minimum a Skeleton and a Parent attribute, or the XML configuration file will be rejected. If a duplicate entry occurs for with the same Skeleton and Parent values, only the first will be added to the cross reference table, the second will be ignored.

### <ImbedRecurseFail>

This is a true/false element. It affects the expansion function. Setting the value to *true* will cause an expansion to halt if a skeleton expansion encounters a previously expanded skeleton. The following code fragments represent a valid run-time use of recursion:

**Skeleton A:**

⁞

)SET VAR = 1

)IM B

⁞

**Skeleton B:**

⁞

)SEL &VAR = 1

)IM C

)ENDSEL

⁞

**Skeleton C:**

⁞

)SET VAR = 2

)IM B

)ENDSEL

⁞

Because the )IM of skeleton C in skeleton B is conditional, this logic causes no problems at run time. The )IM statements work as follows:

A → B → C → B (C is not processed again due to the value of &VAR)

However, the expansion function in the skeleton analysis utility does not know the run-time values of variables. To the expansion utility, the )IM B in skeleton C could cause an infinite recursion:

A → B → C → B → C → B → …

<ImbedRecurseFail>true</ImbedRecurseFail> generates an error when the second )IM of skeleton B (marked red) is encountered.

If <ImbedRecurseFail>false</ImbedRecurseFail> is specified, then the expansion continues, however the expansion will not include the second )IM of skeleton B. Therefore any logic relative to C imbedding B will not be evident.

### <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 Run Time Versions

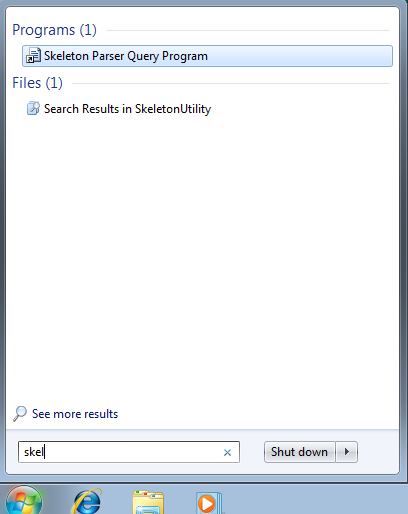
When creating files that will be loaded into relational databases, the ConfigurationNumber or the ConfigurationName can be used to distinguish between the cross-reference data for different PDS concatenations. There must be one configuration XML file for each set PDS concatenation that you wish to define. You must change the ConfigurationNumber and/or ConfigurationName in each XML input file that you process.

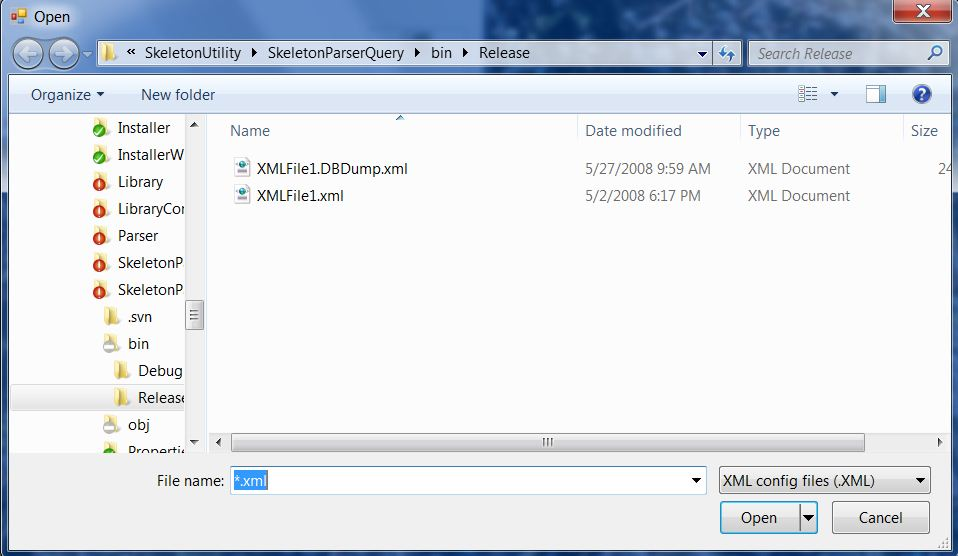
Sample batch .CMD files have been supplied showing how the utility is invoked. The Samples directory also contains files written in REXX showing how to invoke the parser from REXX. You must have a REXX interpreter installed on your system to use these files. The samples were tested with Regina REXX,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

## The Graphical Front-End

The graphical front-end (GFE) can be used to pass an XML configuration file to the parser engine. This configuration file is not required to generate any of the output files (CSV, XML, fixed format) normally created in the batch run of the parser. The GFE invokes the parser engine and then uses the transient intermediate data objects created by the parser as relational data.

When the ISPF skeleton analyzer is installed, the GFE is added to the Start Menu's program items under the name “Skeleton Parser Query Program”:

  
Each time you run the program, it will prompt for a configuration XML file:



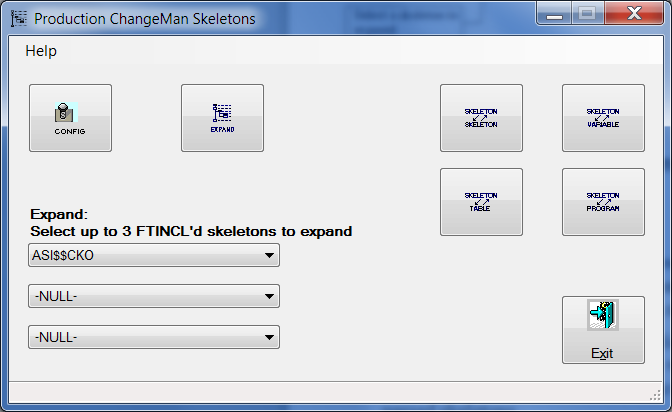
The file you select is validated. Then the skeleton files in the folders specified in the <Directory> entries in the XML file are scanned. The cross-reference information is placed in transient tables, internal to the program.

Following the scan the query program's home dialog is displayed:

<Display> name from the XML configuration file.

Display the Skeleton-to-Variable cross-reference table

Display the Skeleton-to-Skeleton cross-reference table.



Display the Skeleton-to-Table (DOT) cross-reference table

Select a new XML configuration file.

Select one to three skeletons to expand.

Display the Skeleton-to-Executed program cross-reference table.

Expand the skeleton (s) named in the drop-down selection box.

### Load a New Configuration



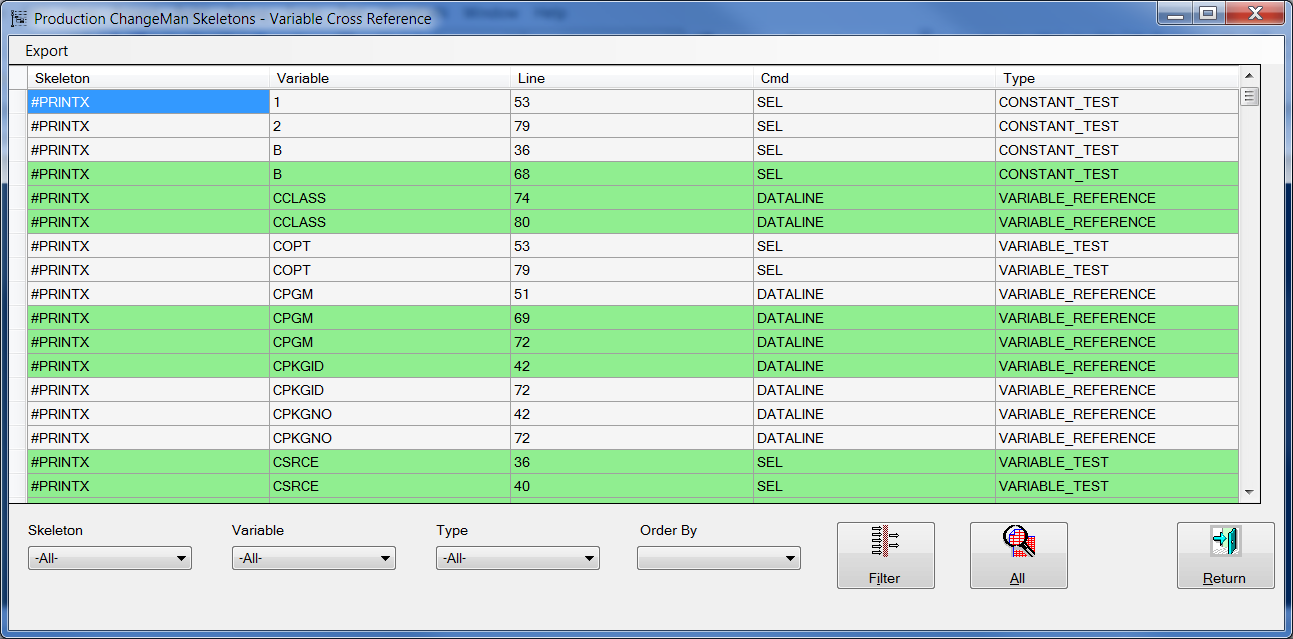
The *New Configuration* button allows you to browse for a new configuration file. The skeletons indicated by the new file will be scanned. Any prior data relationships will be deleted. The internal transient tables will be loaded with new data relationships.

If any of the queries (Skeleton-Skeleton, Skeleton-Variable, Skeleton-Table, Skeleton-Program or the Skeleton Expansion) are open, you will be prompted to either close them and load and parse skeletons from a new configuration or to cancel and return to using the current configuration with the already parsed skeletons.

### Skeleton to Variable Cross-Reference



The Skeleton-Variable button displays the Variable cross-reference Panel:



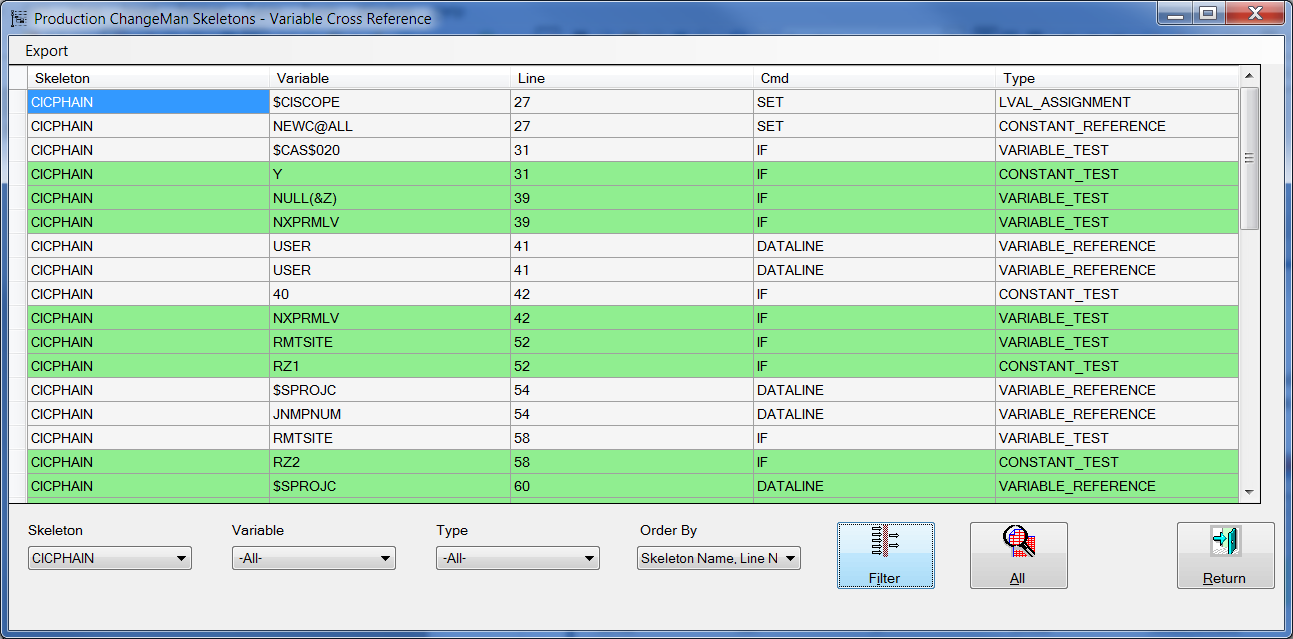
Filters

Sorting

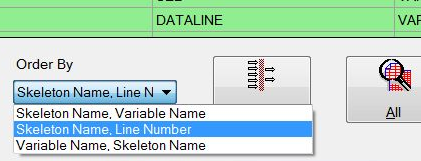
Filter Execution

Filter Discontinue

When the table is first displayed, all variables of all reference types in all skeletons are displayed. You can filter the results by selecting a particular skeleton and/or variable and/or reference type; then push the “Filter” button. Here are the results filtered by a specific skeleton:



Note that the sort order was changed by selecting the Order By drop down and choosing “Skeleton Name, Line Number”. There are three ways the data can be sorted:



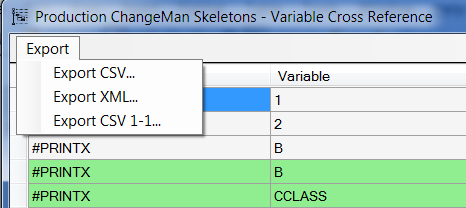
It is not possible to select multiple skeletons or variable names when filtering the data. But you can have multiple instances of the cross-reference dialog open simultaneously. Each can be separately filtered.

To discontinue filtering, simply press the “All” button.

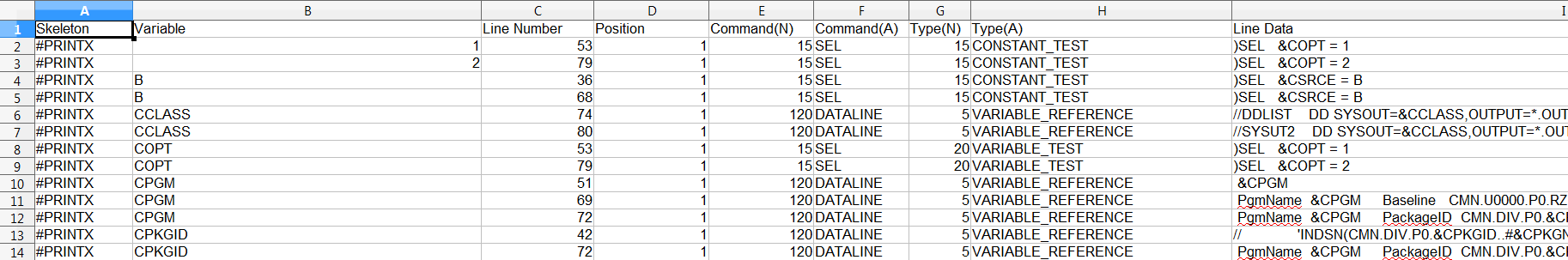
#### Displaying the Skeleton

Double click on any line in the Skeleton/Variable cross-reference grid to open the skeleton and position on the line number where the variable reference occurs.

#### Exporting data



There are three data export functions. The first Export menu item on this form is **Export CSV...** Select the export to create a comma separated value file of the currently active data. This file can be loaded into a spreadsheet or database:



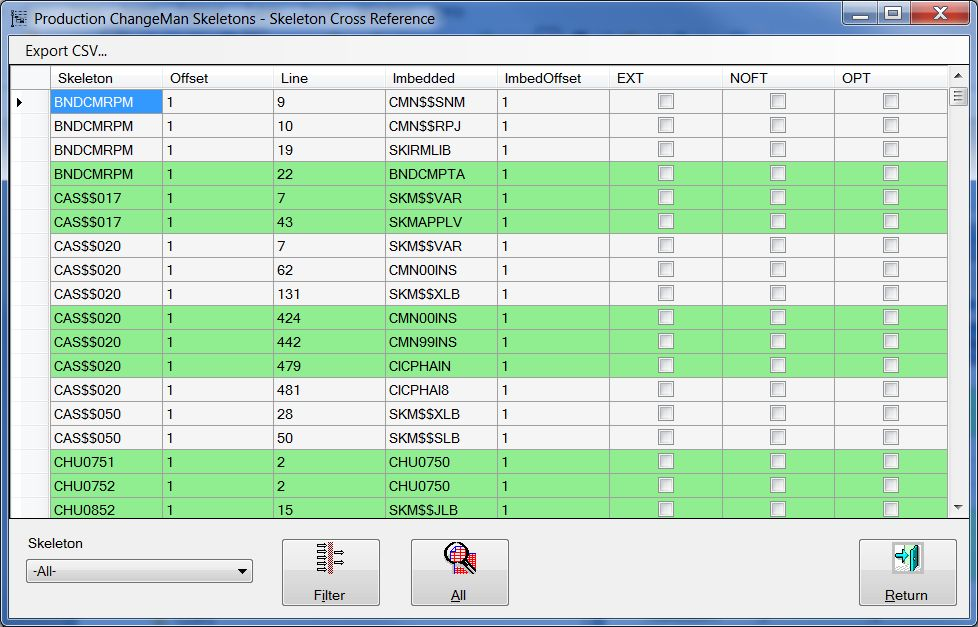
The second Export menu item **Export XML...** is used to export the currently active data as an XML data file.

The last Export menu item **Export CSV 1-1...** is used to export a comma separated value file containing only the skeleton and variable names in a one-to-one relationship. This can allow you to document variable usage within a skeleton. Once you manually update the exported data with your comments, you should not export to that CSV again, or you will wipe out your updates.

### Skeleton to Skeleton Cross-Reference



The Skeleton-Skeleton button displays the Skeleton cross-reference panel:

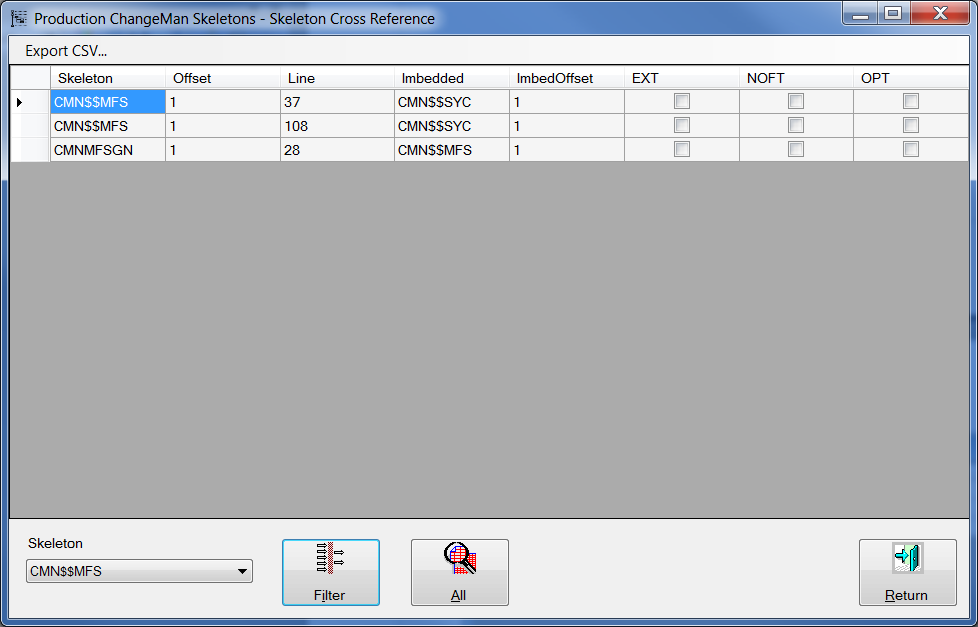


Filter

Filter Execution

Filter Discontinue

When the table is first displayed, all skeletons that imbed other skeletons are shown along with the skeletons they imbed. You can filter the results by selecting a particular skeleton; then push the “Filter” button. Here are the results filtered by a specific skeleton, CMN$$MFS:



Note that the filtered skeleton cross-reference shows all the skeletons that issue )IM for CMN$$MFS as well as those skeletons for which CMN$$MFS issues )IM's.

Columns:

**Skeleton** This is the imbedding (parent) skeleton.

**Offset** The PDS in the concatenation where the member is located.

**Line** The line in the parent skeleton where the )IM statement was found

**Imbedded** The imbedded (child) skeleton

**ImbedOffset** The PDS in the concatenation where the imbedded member is located. If the value is -1, then the imbedded skeleton does not exist in the concatenation.

**EXT** If the )IM statement contains the EXT (extensions) option, this box will be checked

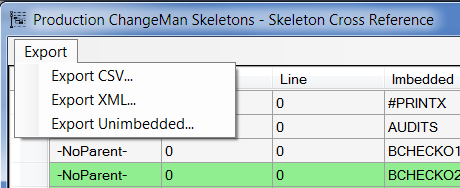
**NOFT** If the )IM statement disables file tailoring (the NOFT option) for the imbedded skeleton, this box will be checked.

**OPT** If the )IM statement indicates that the imbedded skeleton is optional (that is no error will occur if the )IM'ed skeleton does not exist), then this box will be checked.

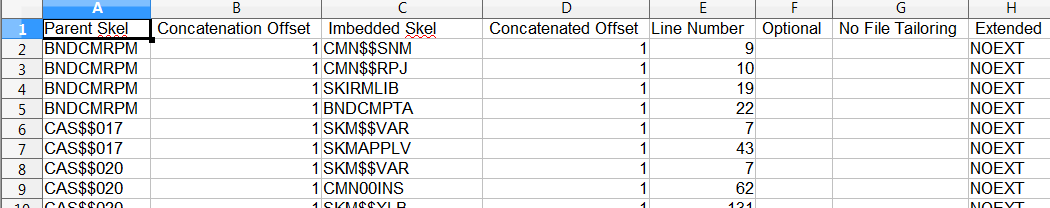
#### Display the Skeleton

Double click on any line in the Skeleton/Skeleton cross-reference grid to open the skeleton and position on the line number where the )IM reference occurs. For imbedded skeletons, the parent or imbedding skeleton will be displayed. For skeletons that are top level skeletons, the top level skeleton will be displayed. In the latter case, since there is no )IM statement involved, the display will be positioned at the first line of the skeleton

#### Exporting data

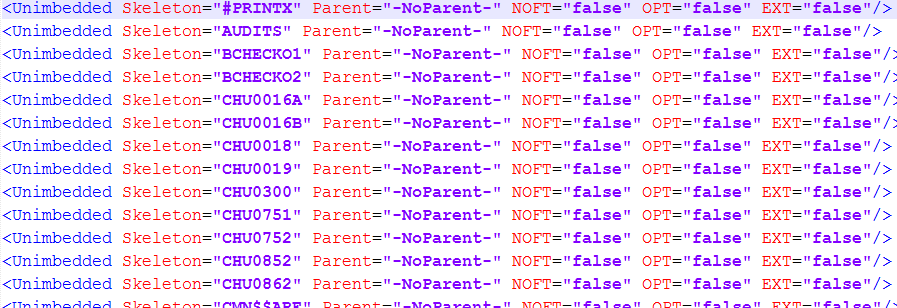


There are three data export functions. The first Export menu item on this form is **Export CSV...** Select the export to create a comma separated value folder of the currently active data. This file can be loaded into a spreadsheet or database:



The second Export menu item **Export XML...** is used to export the currently active data as an XML data file.

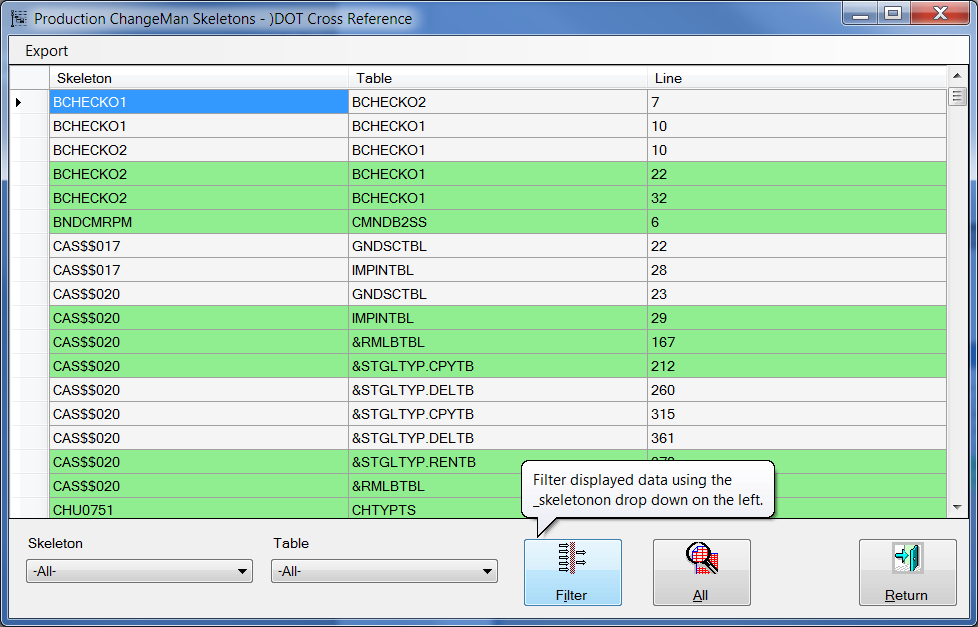
The last Export menu item **Export Unimbedded** exports an set of XML data lines that can be copied into the configuration file, where you can define REXX, z/OS assembly language or a higher level language program or function as the parent for skeletons that are not imbedded by other skeletons, but rather by FTINCL from a program or dialog. The exported data looks like this:



### Skeleton to Table (DOT) cross-reference



The Skeleton-Table button displays the Table cross-reference panel:

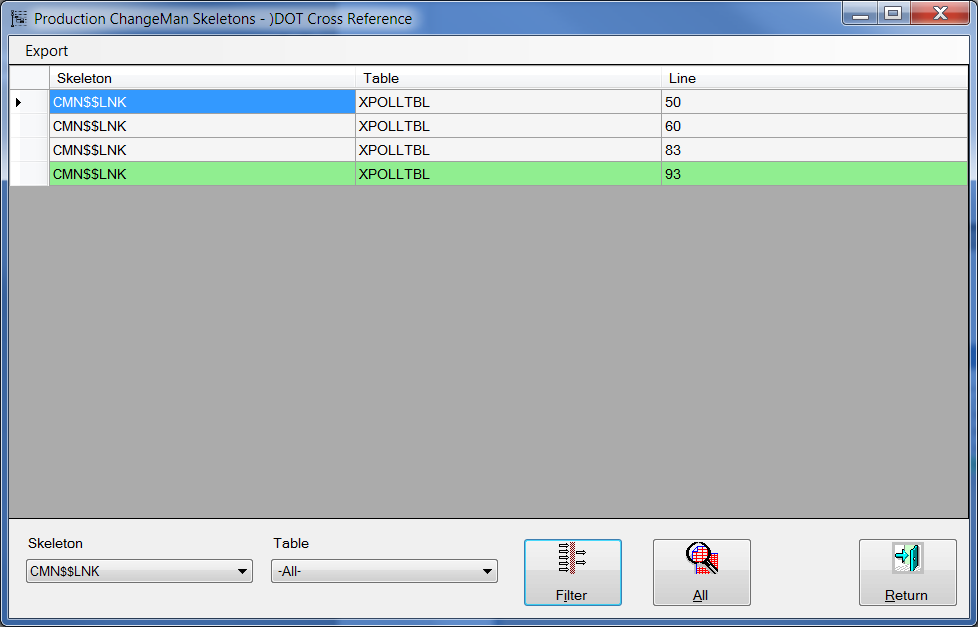


Filter Execution

Filter Discontinue

Filters

When the cross-reference is first displayed, all skeletons and table loops (DOT) are shown. You can filter the results by selecting a particular skeleton and/or table; then push the “Filter” button. Here are the results filtered by a specific skeleton, CMN$$LNK:



Columns:

**Skeleton** This is the imbedding (parent) skeleton.

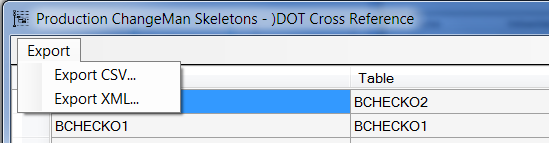
**Table** The table name. Note: Tables with names containing variables are also shown, if they exist.

**Line** The line in the skeleton where the )DOT statement was found

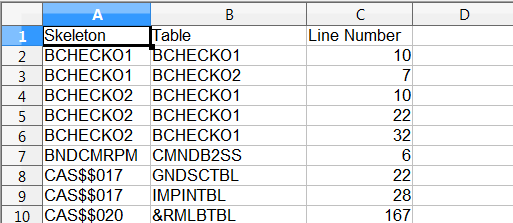
#### Display the Skeleton

Double click on any line in the Skeleton/)DOT cross-reference grid to open the skeleton and position on the line number where the )DOT reference occurs.

#### Exporting data



There are two data export functions. The first Export menu item on this form is **Export CSV...** Select the export to create a comma separated value folder of the currently active data. This file can be loaded into a spreadsheet or database:

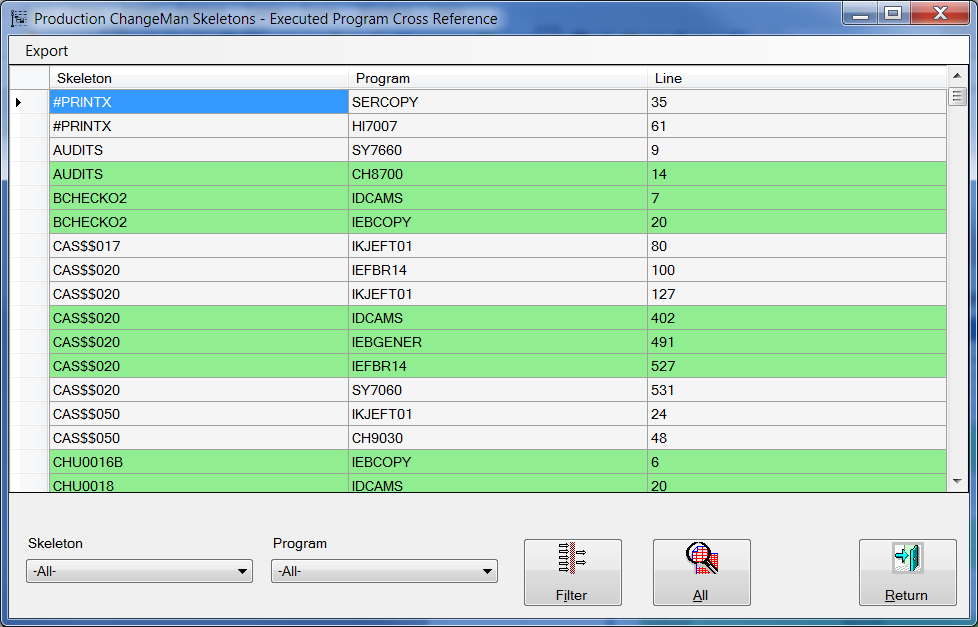


The second Export menu item **Export XML...** is used to export the currently active data as an XML data file.

### Skeleton to Program Cross-Reference



The Skeleton-Program button displays the executed program cross-reference panel:

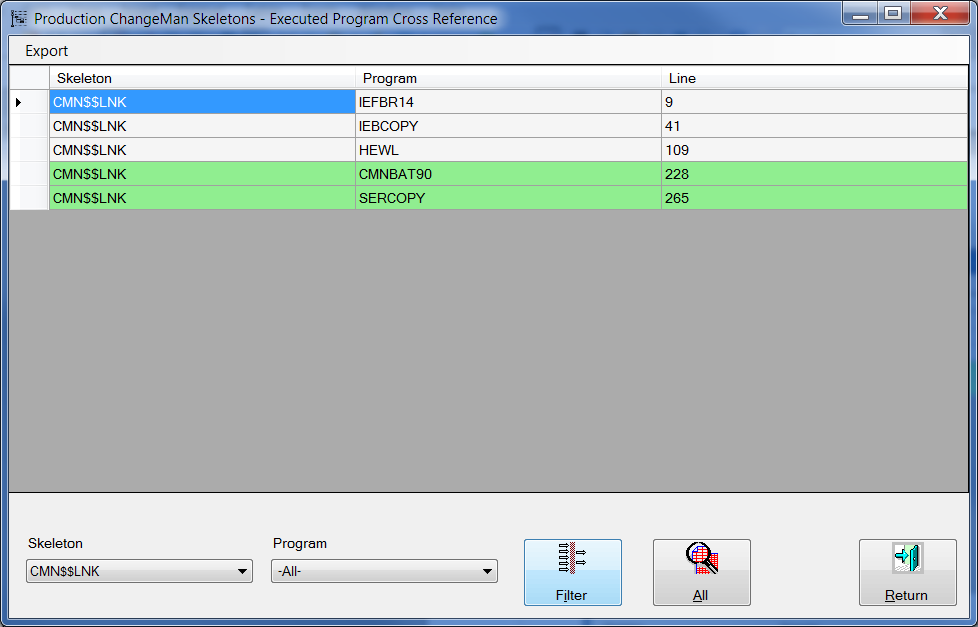


Filter Execution

Filters

Filter Discontinue

When the cross-reference is first displayed, all skeletons and executed program names are shown. You can filter the results by selecting a particular skeleton and/or program name; then push the “Filter” button. Here are the results filtered by a specific skeleton, CMN$$LNK:



Columns:

**Skeleton** This is the imbedding (parent) skeleton.

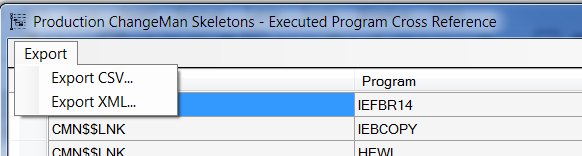
**Program** The executed program name. Note: Programs with names containing variables are also shown, if they exist.

**Line** The line in the skeleton where the )DOT statement was found

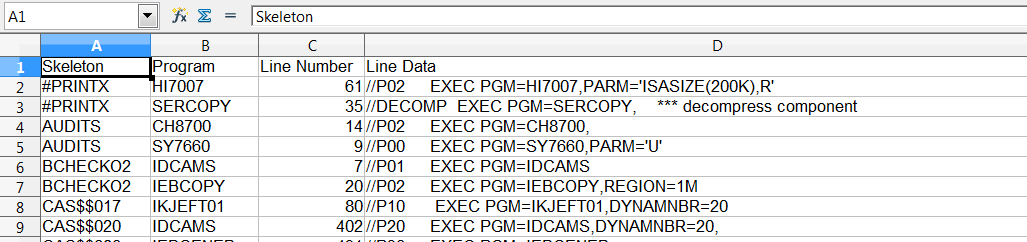
#### Display the Skeleton

Double click on any line in the Skeleton/Executed Program cross-reference grid to open the skeleton and position on the line number where the EXEC PGM reference occurs.

#### Exporting data



There are two data export functions. The first Export menu item on this form is **Export CSV...** Select the export to create a comma separated value folder of the currently active data. This file can be loaded into a spreadsheet or database:



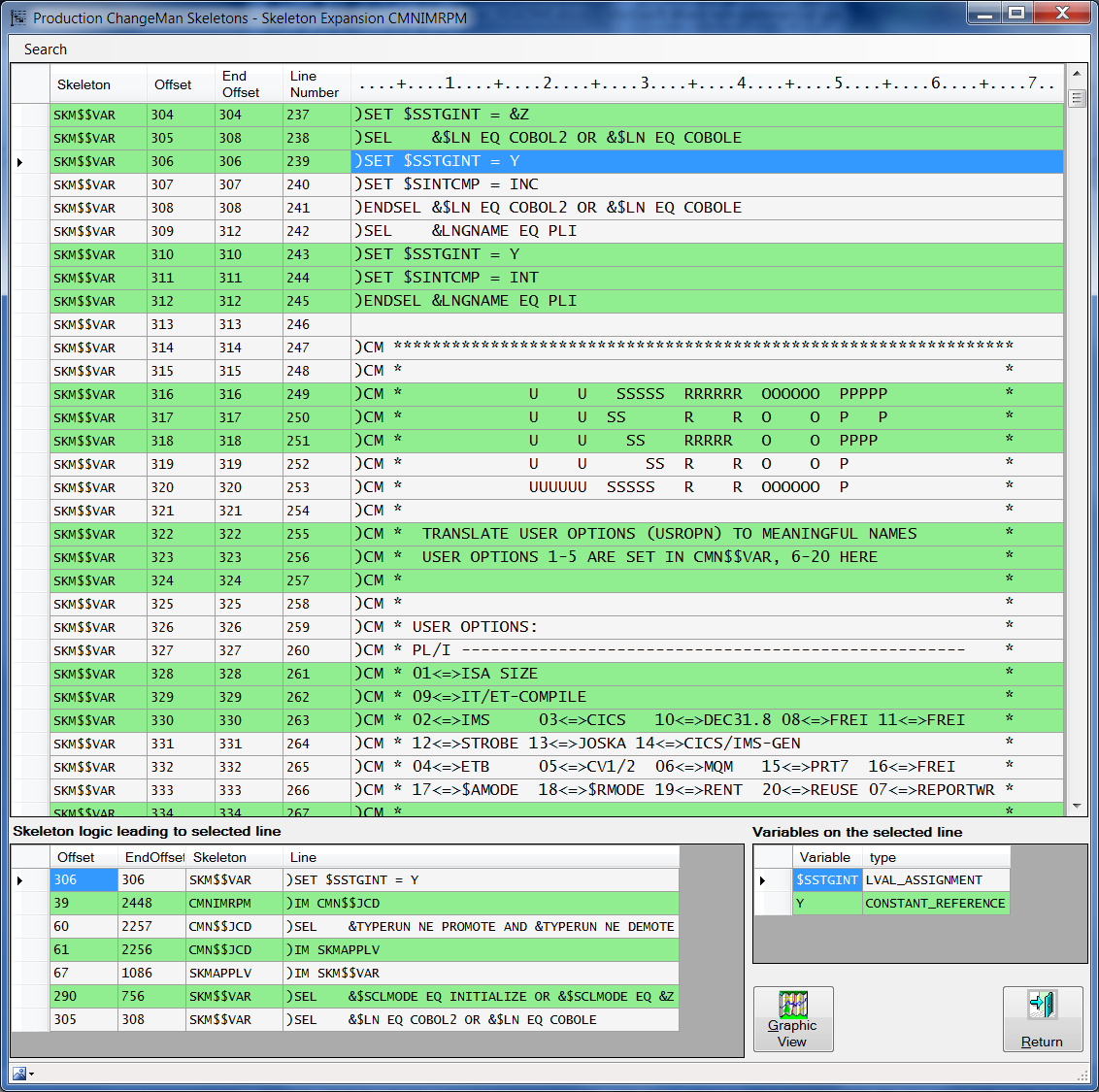
The second Export menu item **Export XML...** is used to export the currently active data as an XML data file.

### Skeleton Expansion



The skeleton expansion function operates on the skeleton indicated in the three drop down selectors on the home form. These drop downs are processed as three separate, sequential FTINCL statements. The expansion processes each skeleton in order and then process each )IM statement in the currently processed FTINCL skeleton, adding the imbedded skeleton to the expansion. When an imbedded skeleton also has an )IM statement, the skeleton associated with that statement is processed.

The result is placed in a viewable, scrollable table:



The selected (current) line.

The skeleton commands that were in effect to reach the selected line.

Variables and/or constants referenced on the highlighted line

**Columns:**

**Skeleton** The skeleton currently being processed.

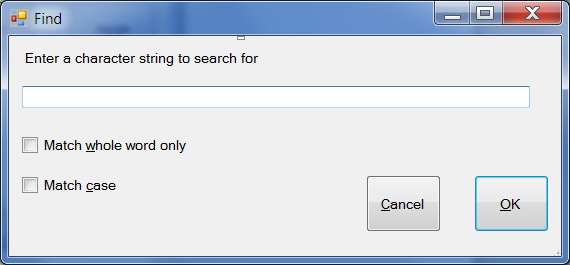
**Offset** The relative offset of this line from the start of the expansion

**LineNo** The offset of the line in the skeleton

This data is followed by the actual line from the skeleton.

#### Expansion Main Menu

The main menu for this function allows you to search for character strings starting at the top line of the expansion. Select the Search menu item and submenu item Find or type Ctl-F.



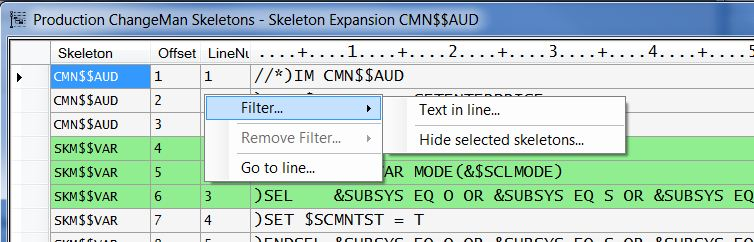
Type the character string you want to find. Choose the “Match whole word only” and/or “Match Case” options. Then press OK to do the search or Cancel to skip the search.

After the first line of text matching your search string is displayed, pressing the F3 function key will find the next occurrence of the string. Continue to press F3 until no more occurrences are found. After the “no more data” box is indicated, pressing F3 once more will reposition on the first line matching your search string.

#### Expansion Context Menu

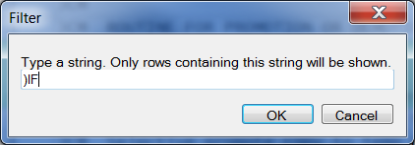
##### Filtering Text Within a skeleton line

The expansion grid has a context menu for filtering the displayed lines or for moving to a specific line in the expansion:

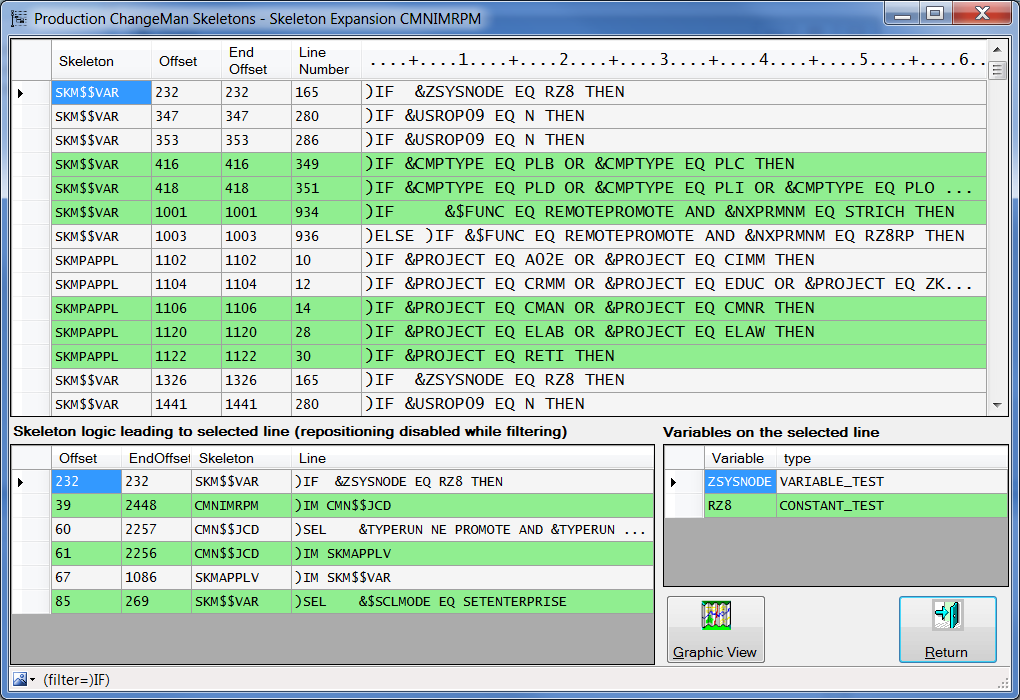


The context menu is activated by positioning the mouse pointer in the main expansion grid and clicking on the right mouse button.

The **Filter...Text in Line...** selection allows you to specify a character string to find:

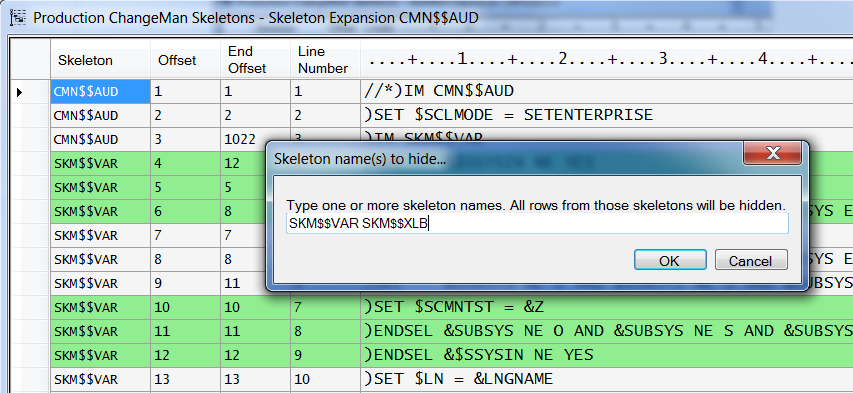


Filtering for )IF finds all lines with )IF somewhere in the text. Then only those lines are displayed:



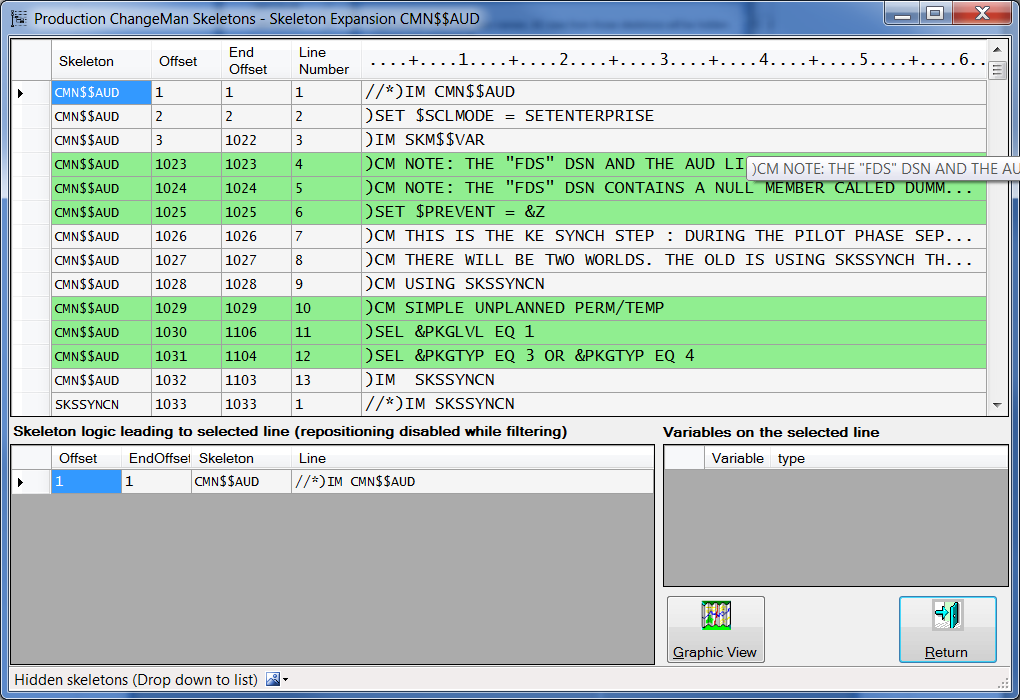
##### Using a Filter to Hide Skeleton Expansions

There is a second filter type, used to hide a skeleton in the expansion. One or more skeletons can be hidden to allow easier analysis of lines:

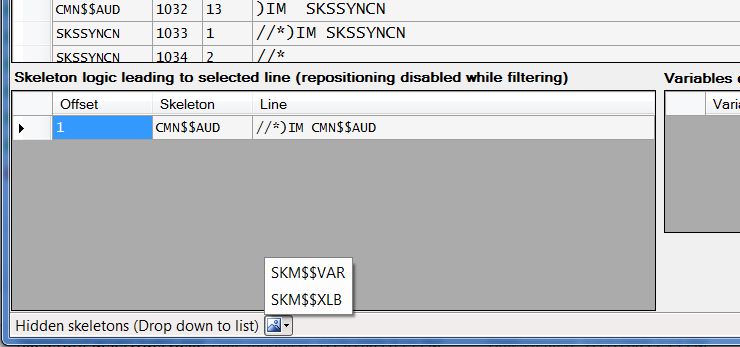


As shown, you may select multiple skeletons in a single request. You can also make multiple requests, hiding more skeletons.

The above filter has the result shown below. Note that after the )IM of SKM$$VAR on line 3, that 1019 lines of SKM$$VAR are suppressed. Also note that in the Status area at the bottom of the form, that there is now a drop down box titled “Hidden skeletons (Drop down to list)”.

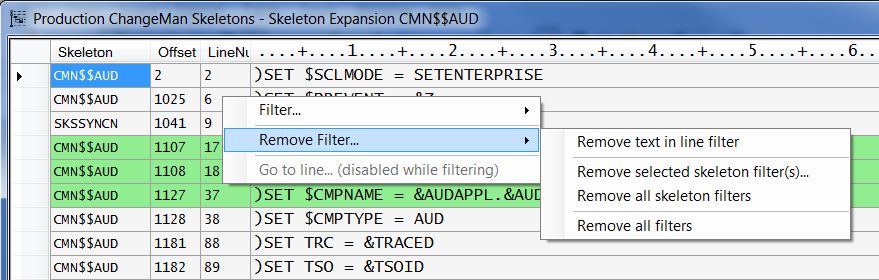


After the drop down is clicked, the list of hidden skeletons is shown.



##### Removing Filters

Once any filter has been applied, the **Remove Filters...** context menu item is enabled:



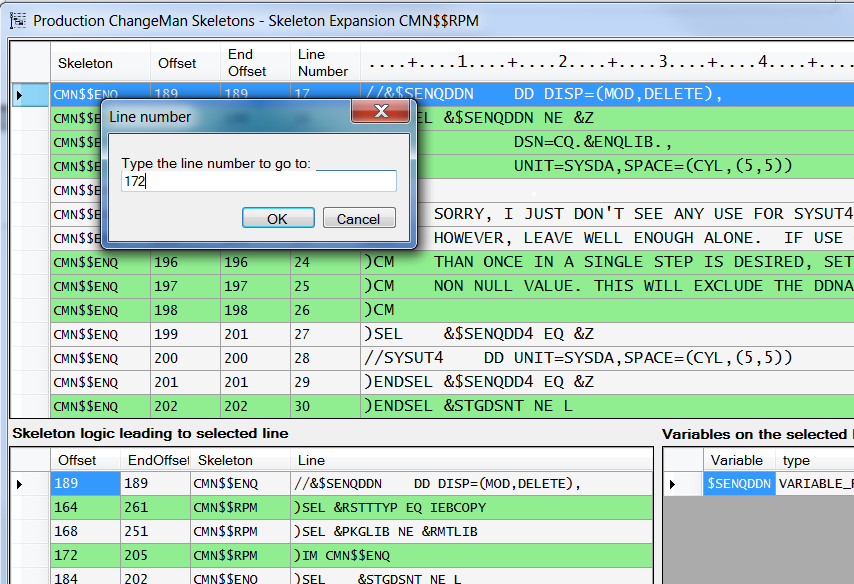
You can remove the active text filter (in this case the filter is on “)SET”). You can also remove any or all skeleton filters. The first skeleton hiding filter removal menu item is for picking one or more hidden skeletons to remove. The next skeleton hiding filter removal menu item redisplays all skeletons.

The “Remove all filters” selection redisplays all skeletons in the expansion, as well as removing the text filter.

There is one additional way to redisplay a hidden skeleton. Using the “Hidden skeletons” drop down list, you can select any of the skeletons in the list. This will redisplay the selected skeleton.

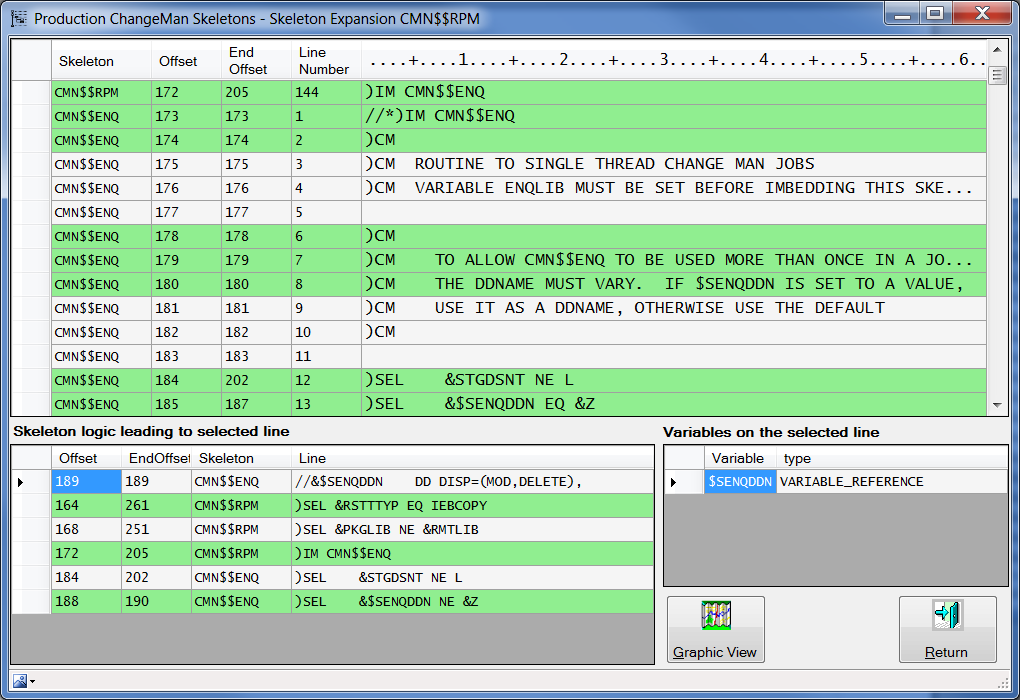
##### Positioning on a Specific Line (Goto)

You can view lines in the expansion grid by clicking on the Offset and End-Offset cells in the Skeleton Logic grid. The expansion grid will display the line referenced in the cell you clicked on. If possible, a few lines above the selected line will also be displayed, so that the line may be a few lines down. You may notice in some cases that the grid will be displayed with the indicated line at the top. You may also use the **Go to line...** context menu item to reposition the grid to a specified. Neither the Go to line... context menu item nor the Skeleton Logic Offset/End-Offset click will change the current row pointer.



Select a line in the list of commands active for the current line (189).

The current line in the expansion will remain 189 and the Skeleton Logic grid will remain unchanged.



Notice that the selected line is still line 189 and that the “Skeleton logic” area has not changed. But the **Go to line...** has positioned the grid so that the selected line is now displayed. In this example, you can return to line 189 either by using the **Go to line...** context menu item and either specifying the line number 189, or by leaving the line number blank, or by simply clicking in the Offset cell in the first line of the Skeleton Logic grid.. The **Go to line...** is not limited to the lines in the logic grid.

To make the **Go to line...** target line the current line, simply click on the line when it is displayed.

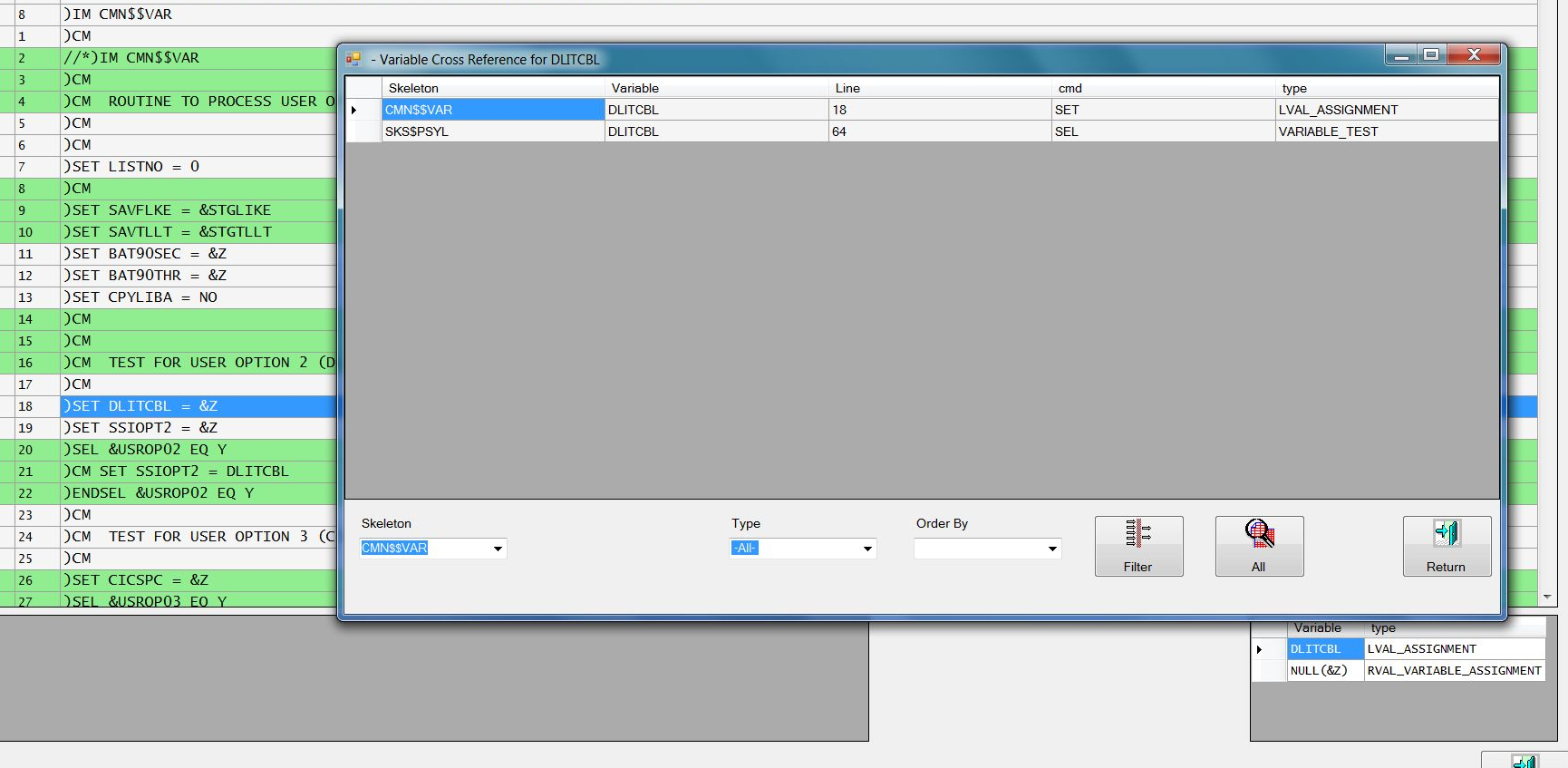
#### Skeleton Logic Display For A Selected Line

When a line is selected by clicking on it the table on the lower left will show the ISPF Skeleton commands: )SEL/IF/ELSE/DO/DOT/IM that were in effect when this line was encountered. The table has the label *Skeleton logic leading to the selected line*. By clicking with the left mouse button in any line within this table the Expansion table view changes to show that line and two lines preceding and the lines that follow the selected line. Changing the view does not change the selected line, so the skeleton logic table remains unchanged. You can quickly return to the selected line by clicking on the top line in the skeleton logic table.

The left mouse button click functionality of the *Skeleton logic...* table is disabled when the line filter is activated in the expansion grid.

#### Variables Display Within Lines

The table on the lower right will show the variables and constants referenced on that line. This table has the label *Variables on the selected line*.If you double click a line in this table, a Skeleton-Variable cross-reference form will open showing all the skeletons that use the variable or constant:



#### Graphic View

On the expansion form there is a button labeled “Graphic View”. Press this button to show the relationships among all of the )IM'd skeletons in an expansion:

