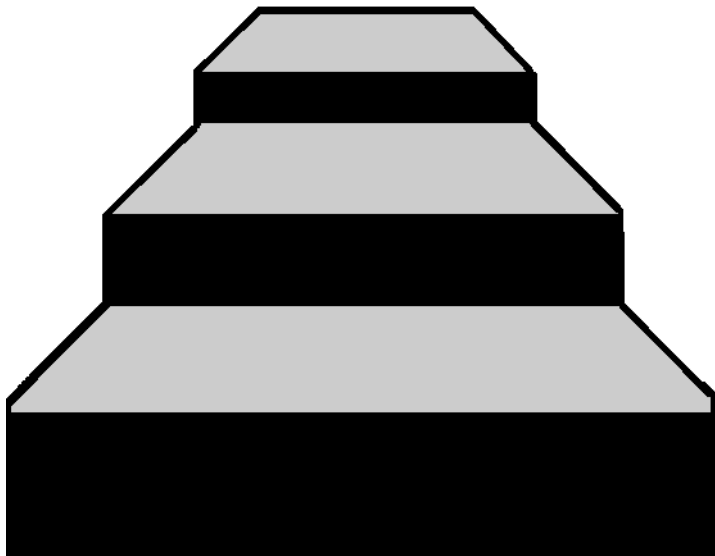




P21 Working Set Library Reference Manual



STEP Tools[®] Software

Contributors

Written by Alexey Lipatov.

© Copyright 1991-2010 STEP Tools, Inc. — All Rights Reserved.

This document contains proprietary and confidential information of STEP Tools, Inc. The contents of this document may not be disclosed to third parties, copied, or duplicated in any form, in whole or in part, without the prior written permission of STEP Tools, Inc.

ST-Developer, ST-Machine, ST-Viewer, and the ST prefix are trademarks of STEP Tools, Inc. STEP Tools is a registered trademark of STEP Tools, Inc. Other brand or product names are the trademarks or registered trademarks of their respective holders.

STEP Tools, Inc.
14 First Street
Troy, New York 12180

Phone (518) 687-2848
Fax (518) 687-4420
E-Mail info@steptools.com
Web <http://www.steptools.com>



Contents

Contents	iii
1 — Introduction	1
1.1 Overview	1
1.2 Example	2
2 — Library Functions	3
2.1 Overview	3
2.2 p21_get_file_info()	3
2.3 p21_get_id_list()	4
2.4 p21_get_type_from_id()	4
2.5 p21_get_references()	5
3 — P21WorkingSet	7
3.1 Overview	7
3.2 P21WorkingSet Constructors	8
3.3 P21WorkingSet Destructor	8
3.4 add()	8
3.5 remove()	9
3.6 isaMode()	9
3.7 setSafeMode()	10
3.8 reset()	10
3.9 findDesign()	10
3.10 useDesign()	11
3.11 findDesignsAs()	11
3.12 useDesignsAs()	12
3.13 saveDesign()	12
3.14 saveDesignAs()	12
3.15 extract()	13

3.16	split()	13
4	— P21FileInfo	15
4.1	Overview	15
4.2	P21FileInfo Data Members	15
4.3	P21FileInfo Constructor	16
4.4	P21FileInfo Destructor	16
5	— P21FileSplitter	17
5.1	Overview	17
5.2	P21FileSplitter Constructors	17
5.3	P21FileSplitter Destructor	18
5.4	add()	18
5.5	remove()	19
5.6	isaMode()	19
5.7	setSafeMode()	19
5.8	reset()	20
5.9	setLeftoverName()	20
5.10	split()	21
6	— Working Set Syntax	23
6.1	Overview	23
6.2	Syntax Specification	23

1 Introduction

1.1 Overview

The Part 21 Working Set library is a ROSE library extension for extracting and processing data from very large STEP files. The library contains several C++ classes and functions that can be used to perform the following operations:

- Obtaining statistical information about a STEP Part 21 file.
- Retrieving entity IDs for a set of entity types.
- Finding types of particular entity instances.
- Finding references of the given entity instances.
- Extracting a subset of entities, called the “working set,” from a file according to your specification.
- Loading the specified working set into the memory as a **RoseDesign** object.
- Merging the modified working set into the original file.
- Splitting a large STEP file into several smaller files, each containing a separately specified subset of entities.

These operations are done “out-of-memory;” the original large file is never loaded into memory as C++ objects. Operations are done on the file as a text stream, and can run quickly. For simple operations on large files, this technique can be faster than reading the entire file into memory. On files less than a few megabytes in size,

or with more complex operations, the ROSE library will outperform the Part 21 Working Set library.

The working set descriptions can be used to cut out some dependency branches. Normally the extraction process will take all dependents of the entities in the working set. Using the extended syntax some of the dependents can be selectively left out. The extended syntax can also be used for excluding particular types, or using entity IDs instead of type names.

1.2 Example

The following example extracts a geometric shape from an AP-203 file given as the first program argument:

```
#include <WorkingSet.h>

void main(int argc, char* argv[]) {
    RoseDesign * design;
    P21WorkingSet WS;

    WS.add("advanced_brep_shape_representation");
    design = WS.useDesign(argv[1]);
    design-> saveAs("shape");
}
```

The first statement creates a working set object. The working set associated with this object is initially empty.

```
P21WorkingSet WS;
```

The next statement adds a single entry to the working set. All entities of the type **advanced_brep_shape_representation** and all entities that they refer to will be loaded into memory by the **useDesign()** function.

```
WS.add("advanced_brep_shape_representation");
design = WS.useDesign(argv[1]);
```

The working set **useDesign()** function uses the same arguments as the **RoseInterface::useDesign** function. The last statement saves the extracted **RoseDesign** object in a new file named “shape.stp.”

```
design-> saveAs("shape");
```

2 Library Functions

2.1 Overview

Many features of the Working Set library are provided through C++ class member functions. However, some services are provided through ordinary functions. These functions are documented in the following sections.

2.2 `p21_get_file_info()`

```
RoseErrorCode p21_get_file_info(  
    P21FileInfo* file_info  
);
```

The function retrieves the file statistics. The `file_info` argument is a pointer to a **P21FileInfo** structure and is used for both input and output. You have to fill the `name` member of this structure with the name of a STEP Part 21 file. The function will fill the rest of the structure with statistical information, such as the number of entities in the file, the number of complex entities, the number of simple entities, and the number of types.

The function returns **ROSE_OK** if the file can be opened, **ROSE_GENERAL** otherwise.

2.3 p21_get_id_list()

```
RoseErrorCode p21_get_id_list(
    char* file_name,
    SetOfString* types,
    SetOfString* ids,
    RoseBoolean isa_mode = ROSE_FALSE
);
```

The **p21_get_id_list()** function obtains the list of instances (entity IDs) for the list of given types. The **types** argument is a pointer to the list of types for which you want to obtain the instances. The **ids** argument is a pointer to the list that receives the list of instances. The **isa_mode** flag specifies the function behavior — if it is set to **ROSE_TRUE** then the function will consider all complex types that have an IS-A relationship to the types given in the **types** argument. If this flag is set to **ROSE_FALSE** (default) the function will only consider exact types.

The function returns **ROSE_OK** if the file can be opened, **ROSE_GENERAL** otherwise. If no instances of the given types are found, no elements are added to the **ids** list.

2.4 p21_get_type_from_id()

```
RoseLogical p21_get_type_from_id(
    char* file_name,
    char* entity_id,
    char* type
);

RoseLogical p21_get_type_from_id(
    char* file_name,
    unsigned long entity_id,
    char* type
);
```

These functions get the type(s) of the specified entity. The entity ID is given in the **entity_id** argument in the form of a “#<number>” string for the first implementation of this function or in the numeric form for the second implementation. The entity type is returned in the **type** argument that should point to a string long enough to accommodate the type name.

The function returns **ROSE_FALSE** if the file can not be opened, **ROSE_TRUE** if the given ID is for a single type, **ROSE_UNKNOWN** if the given ID is for a set of types

(complex entity). In the last case the list of types will be in the type argument, separated by spaces.

2.5 p21_get_references()

```
RoseErrorCode p21_get_references (
    char* file_name,
    char* entity_id,
    SetOfint* refs
);
```

```
RoseErrorCode p21_get_references (
    char* file_name,
    unsigned long entity_id,
    SetOfint* refs
);
```

This function retrieves the list of references for the given entity (the list of entity instances this entity refers to). The entity ID is given in the **entity_id** argument in the form of a “#<number>” string for the first implementation of this function or in the numeric form for the second implementation. The references are returned as a list of integers pointed to by the **refs** argument.

The **p21_get_references** function returns **ROSE_OK** if the file can be opened, **ROSE_GENERAL** otherwise. If the given entity does not contain any references, no elements are added to the **refs** list.

3 P21WorkingSet

3.1 Overview

The **P21WorkingSet** class is used to either extract entities included in the working set from a STEP file or to split the file into two parts — one containing all the entities in the working set and the other containing the rest of the entities from the original file.

The class has methods to obtain a **RoseDesign** object representing the design with the entities included in the working set. It also has methods that manipulate STEP files directly generating new STEP files with extracted and/or discarded entities.

The class also provides methods to manipulate the working set itself by adding and removing entities.

The working set can be specified using the extended syntax to cut out some reference chains. Normally the extraction process will take all references of the entities in the working set. Using the extended syntax some of the references can be selectively left out. The extended syntax can also be used for excluding particular types, or using entity IDs instead of type names.

3.2 P21WorkingSet Constructors

```
P21WorkingSet (
    RoseBoolean isa_mode = ROSE_FALSE
);
```

This constructor creates an empty working set object. If the **isa_mode** argument is true, the object will consider complex (**AND/OR**) entities as having an IS-A relationship to entities in the working set. A complex entity does not have a specific type, but rather has a set of types. When this argument is **ROSE_TRUE** the object will take a complex entity if at least one of its types is present in the working set. If this argument is **ROSE_FALSE**, the object will consider only exact types, thus ignoring complex entities altogether.

```
P21WorkingSet (
    STR ws_file_name,
    RoseBoolean isa_mode = ROSE_FALSE
);
```

This constructor creates an object and fills the working set from the given file.

```
P21WorkingSet (
    SetOfString* ws,
    RoseBoolean isa_mode=ROSE_FALSE
);
```

This constructor creates an object and fills the working set from the given list of strings.

3.3 P21WorkingSet Destructor

```
~P21WorkingSet ();
```

The destructor calls the **reset()** function during object clean-up.

3.4 add()

```
void add(
    STR entity
);
```

This method adds a new entity to the working set.

The **entity** argument specifies the entity type or instance ID. For complete syntax specification see Working Set Syntax.

3.5 **remove()**

```
void remove(
    STR entity
);
```

The **remove** method removes the entity from the working set. The **entity** argument is a string specification of the entity. It should match exactly the specification in the working set.

3.6 **isaMode()**

```
RoseBoolean isaMode();

void isaMode(
    RoseBoolean isa_mode
);
```

The **isaMode()** functions get or set the “IS-A” mode flag.

The “IS-A” mode is used for type selection — if this mode is set to **ROSE_TRUE**, the object will consider complex (**AND/OR**) entities that have an IS-A relationship to the entities in the working set. A complex entity does not have a specific type, but rather has a set of types. When the “IS-A” mode is on the object will take a complex entity if at least one of its types is present in the working set. If the mode is reset (**ROSE_FALSE**), the object will consider only exact types, thus ignoring complex entities altogether.

3.7 setSafeMode()

```
void setSafeMode(
    EMODE safe_mode
);
```

This method is used to set the “safety level” of operation. The **safe_mode** argument is an enum value defined as:

```
typedef enum { FAST, SAFE, SAFEST } EMODE;
```

When the safety level is set to **FAST**, the object is configured for maximum performance but some rare syntax in the Part 21 file may cause the object to behave incorrectly. Examples of such syntax include constructs that look like entity references (**#<number>**) inside character strings and comments within entity definitions. For example:

```
#10=MY_ENTITY('Name', 12, #40, 'Will break #130 here', 99.34);
#20=ANOTHER_ONE('Untitled', /* Will break here #50 too */ 8, #100, #70);
```

The entity #10 will cause incorrect operation in the **FAST** mode, but will be processed correctly in the **SAFE** and the **SAFEST** modes. The entity #20 will be processed correctly only in the **SAFEST** mode.

Note — The default setting is always **FAST**.

3.8 reset()

```
void reset();
```

This method cleans the object by removing all entries from the working set.

3.9 findDesign()

```
RoseDesign* findDesign(
    STR design_name
);
```

This method is an equivalent of **RoseInterface::findDesign**. It takes the design file

name as an argument and returns a pointer to a **RoseDesign** object. The resulting design will only contain entities specified by the working set.

Note — the design name should be a name of a STEP Part 21 file, files in ROSE format are not supported.

3.10 useDesign()

```
RoseDesign* useDesign(
    STR design_name
);
```

This method is an equivalent of **RoseInterface::useDesign**. It takes the design file name as an argument and returns a pointer to a **RoseDesign** object. It also makes the resulting design the current design. The resulting design will only contain entities specified by the working set.

Note — the design name should be a name of a STEP Part 21 file, files in ROSE format are not supported.

3.11 findDesignsAs()

```
RoseDesign* findDesignsAs(
    SetOfString* design_names,
    STR design_name
);
```

This method is a merger function. It filters several Part 21 files at once, creating a combined design in memory.

The **design_names** argument is a set of input file names. The **design_name** argument is the name of the new combined design.

3.12 useDesignsAs()

```
RoseDesign* useDesignsAs (  
    SetOfString* design_names,  
    STR design_name  
);
```

This method performs as **findDesignsAs** with the only difference being that the new design will become the current rose design.

3.13 saveDesign()

```
void saveDesign(  
    RoseDesign* design = NULL  
);  
  
void saveDesign(  
    STR design_name  
);
```

This **saveDesign()** function saves the design into the original file. This function merges the given design (or the current rose design if NULL is given) into the original file. The integrity of the original file is preserved.

The second version is the same as the first, but takes the design name instead of the pointer.

3.14 saveDesignAs()

```
void saveDesignAs (  
    STR new_name,  
    RoseDesign* design = NULL  
);  
  
void saveDesignAs (  
    STR new_name,  
    STR design_name  
);
```

These methods are similar to the **saveDesign** methods but save the design under the new name.

3.15 extract()

```
void extract(
    STR design_name,
    STR to_name = NULL
);
```

This method bypasses ROSE completely. It is useful when your application must process files without actually using any design objects. This method is similar to the **findDesign** method, but copies the extracted entities to the file with the name given in the **to_name** argument without loading any design in memory.

If the **to_name** argument is NULL, the new name will be **<design_name>+ws.stp** (Given a name of “my_design”, the result will be “my_design+ws.stp”).

```
void extract(
    SetOfString* design_names,
    STR to_name
);
```

This method is an extension of the first **extract** method to use on a set of files rather than a single file.

3.16 split()

```
void split(
    STR design_name,
    STR ws_name = NULL,
    STR compl_name = NULL
);
```

This method splits the file into a working set file and a complement file. The working set file contains all entities extracted according to the working set specification. The complement file contains all the entities of the original file not included in the working set file.

The **design_name** argument is the name of the Part 21 file to split. The **ws_name** argument is the name of the working set file. If this argument is NULL, the new name will be **<design_name>+ws.stp** (e.g. if the design_name is “my_design”, the resulting working set file name will be “my_design+ws.stp”). The **compl_name** argument is the name of the complement file. If this argument is NULL, the new name will be **<design_name>-ws.stp** (e.g. if the design_name is “my_design”, the resulting complement file name will be “my_design-ws.stp”).

4 P21FileInfo

4.1 Overview

The **P21FileInfo** structure is used by the **p21_get_file_info** function to fill the statistical information obtained from a STEP Part 21 file.

4.2 P21FileInfo Data Members

`STR name;`

This member is the name of the file. This member should be set before calling the **p21_get_file_info** function.

`unsigned long num_entities;`

This member represents the total number of entities in the file. It is being set by the **p21_get_file_info** function.

`unsigned long num_simple;`

This member is a number of simple type entities. (Set by **p21_get_file_info**.)

`unsigned long num_complex;`

This member is a number of complex (**AND/OR**) type entities. (Set by

p21_get_file_info.)

```
unsigned long num_types;
```

This member is a total number of entity types found in the file. (Set by **p21_get_file_info.**)

```
CArrayOfTypeCount histogram;
```

This member represents a histogram of types. It is an array of **TypeCount** structures. A **TypeCount** structure has two members:

```
int m_nCount;           // Number of type instances
STR m_szType;          // Type name
```

4.3 P21FileInfo Constructor

```
P21FileInfo();
```

The constructor creates an object with all counts set to zero.

4.4 P21FileInfo Destructor

```
~P21FileInfo();
```

The destructor removes all entries from the **histogram** member and then destroys the object itself.

5 P21 FileSplitter

5.1 Overview

The **P21FileSplitter** class is used to split a STEP Part 21 file into several parts — each containing the entities in the corresponding working set. The class uses a set of **P21WorkingSet** objects, with one **P21WorkingSet** for each of the split files. The rest of the entities in the original file is written into a special “leftover” file.

5.2 P21FileSplitter Constructors

```
P21FileSplitter(  
    RoseBoolean isa_mode = ROSE_FALSE  
);
```

This constructor implementation creates an empty working set object. The **isa_mode** argument sets the “IS-A” mode for selection — if it is **ROSE_TRUE**, the object will consider complex (**AND/OR**) entities that have an IS-A relationship to entities in the working set. A complex entity does not have a specific type, but rather has a set of types. When this argument is **ROSE_TRUE** the object will take a complex entity if at least one of its types is present in the working set. If this argument is **ROSE_FALSE**, the object will consider only exact types, thus ignoring complex entities altogether.

```

P21FileSplitter(
    STR ws_file_name,
    RoseBoolean isa_mode = ROSE_FALSE
);

```

This constructor creates an object and fills the working set from the given file. This working set file has a different syntax structure than a working set file used by the **P21WorkingSet** class — each entity specification is followed by a semicolon and a split file name into which the entities satisfying this specification should be placed.

```

P21FileSplitter(
    SetOfString* ws,
    RoseBoolean isa_mode=ROSE_FALSE
);

```

This constructor creates an object and fills the working set from the given list of strings. The strings in the set pointed by the **ws** argument should be specified using the same syntax as described above.

5.3 P21FileSplitter Destructor

```

~P21FileSplitter();

```

The destructor cleans the object (calls the **reset** method) and then destroys the object itself.

5.4 add()

```

void add(
    P21WorkingSetPtr* entry,
    STR fileName
);

```

This method adds a new entry to the set of working sets.

The **entry** argument is a pointer to a **P21WorkingSet** object that should be associated with the file name given in the second argument.

5.5 remove()

```
void remove(
    int index
);
```

The **remove** method removes the entry from the set of working sets. The **index** argument is a zero-based index of a working set.

5.6 isaMode()

```
RoseBoolean isaMode();
```

The method returns the “IS A” mode flag.

```
void isaMode(
    RoseBoolean isa_mode
);
```

This method sets the “IS A” mode flag.

The “IS-A” mode is used for type selection — if this mode is set to **ROSE_TRUE**, the object will consider complex (**AND/OR**) entities that have an IS-A relationship to entities in the working set. A complex entity does not have a specific type, but rather has a set of types. When the “IS-A” mode is on the object will take a complex entity if at least one of its types is present in the working set. If the mode is reset (**ROSE_FALSE**), the object will consider only exact types, thus ignoring complex entities altogether.

5.7 setSafeMode()

```
void setSafeMode(
    P21WorkingSet::EMODE safe_mode
);
```

This method is used to set the “safety level” of operation. The **safe_mode** argument is an enum value defined in the **P21WorkingSet** class as:

```
typedef enum { FAST, SAFE, SAFEST } EMODE;
```

When the safety level is set to **FAST**, the object is configured for maximum performance but some rare syntax in the Part 21 file may cause the object to behave incorrectly. Examples of such syntax include constructs that look like entity references (**#<number>**) inside character strings and comments within entity definitions. For example:

```
#10=MY_ENTITY('Name', 12, #40, 'Will break #130 here', 99.34);
#20=ANOTHER_ONE('Untitled', /* Will break here #50 too */ 8, #100, #70);
```

The entity #10 will cause incorrect operation in the **FAST** mode, but will be processed correctly in the **SAFE** and the **SAFEST** modes. The entity #20 will be processed correctly only in the **SAFEST** mode.

Note — The default setting is always **FAST**.

5.8 reset()

```
void reset();
```

This method cleans the object by removing all entries from the set of working sets. It also calls the **reset** method for each of the working sets before removing it.

5.9 setLeftoverName()

```
void setLeftoverName(
    STR szName
);
```

This method sets the name of the “leftover” file. This file is a last result of the split operation — all entities not used by any of the working sets are written to this file. The default name of this file is “leftover.stp”.

5.10 split()

```
void split(  
    STR file_name  
);
```

This method performs the splitting. The `file_name` argument specifies the name of the STEP Part 21 file to split.

Note — This method works only on STEP Part21 files. It will not work on ROSE files.

6 Working Set Syntax

6.1 Overview

The syntax of the working set specification used by the WorkingSet library is similar to the working set syntax used by EXPRESS compiler with the following extensions and limitations:

- It allows you to specify entities by their ID numbers, not only by types.
- It allows you to selectively cut the entity reference chains.
- It does not use the ANDOR constructs (complex types handling is controlled by the **P21WorkingSet** class).

6.2 Syntax Specification

The following special characters are used:

Character	Meaning	Example
<code>;</code>	Comment	<code>; Working set file</code> (Ignores this line)

Character	Meaning	Example
+ (optional)	Include this entity	+my_entity (Includes my_entity)
-	Exclude this entity and all its references	-my_entity (Excludes my_entity and all entities it refers to)
~	Exclude this entity only	~my_entity (Excludes my_entity but retains all its references)
#	Use entity instance ID instead of type	#130 (Includes entity #130)
:	Separates entity specification and file name (P21FileSplitter class only)	#1780: first.stp (When splitting, places entity #1780 into the first.stp file)