%MktOrth Macro

The %MktOrth autocall macro lists some of the 100% orthogonal main-effects plans that the %MktEx macro can generate. See page 106 for an example of using this macro in the design chapter. Also see the following pages for examples of using this macro in the discrete choice chapter: 342 and 660. Additional examples appear throughout this chapter.

Mostly, you use this macro indirectly; it is called by the %MktEx macro. However, you can directly call the %MktOrth macro to see what orthogonal designs are available and decide which ones to use. The following step requests all the designs in the catalog with 100 or fewer runs and two-level through six-level factors (with no higher-level factors.)

```
%mktorth(maxn=100, maxlev=6)
```

The macro creates data sets and displays no output except the following notes:

```
NOTE: The data set WORK.MKTDESLEV has 347 observations and 9 variables.
NOTE: The data set WORK.MKTDESCAT has 347 observations and 3 variables.
```

This next step generates the entire catalog of 119,852 designs* including over 62,000 designs in 512 runs that are not generated by default:

```
%mktorth(maxlev=144, options=512)
```

This step might take on the order of several minutes to run.

This next step generates the catalog of approximately 57 thousand designs including designs with up to 144-level factors:

```
%mktorth(maxlev=144)
```

This step might take on the order of several minutes to run. Unless you really want to see all of the designs, you can make the %MktOrth macro run much faster by specifying smaller values for range= or maxn= (which control the number of runs) and maxlev= (which controls the maximum number of factor levels and the number of variables in the MKTDESLEV data set) than the defaults (range=n le 1000, maxn=1000, maxlev=50). The maximum number of levels you can specify is 144.

The following step lists the first few and the last few designs in the catalog:

```
proc print data=mktdeslev(where=(n le 12 or n ge 972));
   var design reference;
   id n; by n;
   run;
```

*Elsewhere in this chapter, the size of the orthogonal array catalog is reported to be 117,556. The discrepancy is due to the 2296 designs that are explicitly in the catalog and have more than 513 runs. Most are constructed from the parent array 24^8 in 576 runs (which is useful for making Latin Square designs). The rest are constructed from Hadamard matrices.

Some of the results are as follows:

n	Design	Reference
4	2 ** 3	Hadamard
6	2 ** 1 3 ** 1	Full-Factorial
8	2 ** 7	Hadamard
	2 ** 4 4 ** 1	Fractional-Factorial
9	3 ** 4	Fractional-Factorial
10	2 ** 1 5 ** 1	Full-Factorial
12	2 ** 11	Hadamard
	2 ** 4 3 ** 1	Orthogonal Array
	2 ** 2 6 ** 1	Orthogonal Array
	3 ** 1 4 ** 1	Full-Factorial
972	2 **971	Hadamard
976	2 **975	Hadamard
	2 **972 4 ** 1	Orthogonal Array
	2 **969 4 ** 2	Orthogonal Array
	2 **968 8 ** 1	Orthogonal Array
	2 **966 4 ** 3	Orthogonal Array
984	2 **983	Hadamard
	2 **980 4 ** 1	Orthogonal Array
992	2 **991	Hadamard
	2 **988 4 ** 1	Orthogonal Array
	2 **985 4 ** 2	Orthogonal Array
	2 **984 8 ** 1	Orthogonal Array
	2 **982 4 ** 3	Orthogonal Array
	2 **979 4 ** 4	Orthogonal Array
	2 **976 4 ** 5	Orthogonal Array
	2 **976 16 ** 1	Orthogonal Array
	2 **973 4 ** 6	Orthogonal Array
	2 **970 4 ** 7	Orthogonal Array
1000	2 **999	Hadamard
	2 **996 4 ** 1	Orthogonal Array

In most ways, the catalog stops at 513 runs. The exceptions include: 24^8 in 576 runs, Hadamard designs up to $n = 1000$, and designs easily constructed from those Hadamard designs (by creating, 4, 8, 16, and so on level factors).

The following step displays the first few designs and variables in the MKTDESLEV data set:

```
proc print data=mktdeslev(where=(n le 12));
   var design reference x1-x6;
   id n; by n;
   run;
```

Some of the results are as follows:

n	Design	Reference	x1	x2	x3	x4	x5	x6
4	2 ** 3	Hadamard	0	3	0	0	0	0
6	2 ** 1 3 ** 1	Full-Factorial	0	1	1	0	0	0
8	2 ** 7	Hadamard	0	7	0	0	0	0
	2 ** 4 4 ** 1	Fractional-Factorial	0	4	0	1	0	0
9	3 ** 4	Fractional-Factorial	0	0	4	0	0	0
10	2 ** 1 5 ** 1	Full-Factorial	0	1	0	0	1	0
12	2 ** 11	Hadamard	0	11	0	0	0	0
	2 ** 4 3 ** 1	Orthogonal Array	0	4	1	0	0	0
	2 ** 2 6 ** 1	Orthogonal Array	0	2	0	0	0	1
	3 ** 1 4 ** 1	Full-Factorial	0	0	1	1	0	0

If you just want to display a list of designs, possibly selecting on n, the number of runs, you can use the MKTDESCAT data set. However, if you would like to do more advanced processing, based on the numbers of levels of some of the factors, you can use the `outlev=mktdeslev` data set to select potential designs. You can look at the level information in MKTDESLEV and see the number of two-level factors in `x2`, the number of three-level factors in `x3`, ..., the number of fifty-level factors is in `x50`, ..., and the number of 144-level factors in `x144`. The number of one-level factors, `x1`, is always zero, but `x1` is available so you can make arrays (for example, `array x[50]`) and have `x[2]` refer to `x2`, the number of two-level factors, and so on.

Say you are interested in the design $2^5 3^5 4^1$. The following steps display some of the ways in which it is available:

```
%mktorth(maxn=100)

proc print data=mktdeslev noobs;
   where x2 ge 5 and x3 ge 5 and x4 ge 1;
   var n design reference;
   run;
```

Some of the results are as follows:

n	Design	Reference
72	2 ** 44 3 ** 12 4 ** 1	Orthogonal Array
72	2 ** 43 3 ** 8 4 ** 1 6 ** 1	Orthogonal Array
72	2 ** 37 3 ** 13 4 ** 1	Orthogonal Array
72	2 ** 36 3 ** 9 4 ** 1 6 ** 1	Orthogonal Array
72	2 ** 35 3 ** 12 4 ** 1 6 ** 1	Orthogonal Array
.		
.		
.		

The following steps illustrate one way that you can see all of the designs in a certain range of sizes:

```
%mktorth(range=12 le n le 20)

proc print; id n; by n; run;
```

The results are as follows:

n	Design	Reference
12	2 ** 11	Hadamard
	2 ** 4 3 ** 1	Orthogonal Array
	2 ** 2 6 ** 1	Orthogonal Array
	3 ** 1 4 ** 1	Full-Factorial
14	2 ** 1 7 ** 1	Full-Factorial
15	3 ** 1 5 ** 1	Full-Factorial
16	2 ** 15	Hadamard
	2 ** 12 4 ** 1	Fractional-Factorial
	2 ** 9 4 ** 2	Fractional-Factorial
	2 ** 8 8 ** 1	Fractional-Factorial
	2 ** 6 4 ** 3	Fractional-Factorial
	2 ** 3 4 ** 4	Fractional-Factorial
	4 ** 5	Fractional-Factorial
18	2 ** 1 3 ** 7	Orthogonal Array
	2 ** 1 9 ** 1	Full-Factorial
	3 ** 6 6 ** 1	Orthogonal Array

```
 20    2 ** 19                                    Hadamard
       2 **  8          5 **  1                   Orthogonal Array
       2 **  2         10 **  1                   Orthogonal Array
                         4 **  1   5 **  1        Full-Factorial
```

The %MktOrth macro can output the lineage of each design, which is the set of steps that the %MktEx macro uses to create it. The following steps illustrate this option:

```
%mktorth(range=n=36, options=lineage)

proc print noobs;
   where index(design, '2 ** 11') and index(design, '3 ** 12');
   run;
```

The results are as follows:

```
 n          Design             Reference

36    2 ** 11  3 ** 12    Orthogonal Array

                          Lineage

36 ** 1 : 36 ** 1 > 3 ** 12 12 ** 1 : 12 ** 1 > 2 ** 11
```

The design $2^{11}3^{12}$ in 36 runs starts out as a single 36-level factor, 36^1. Then 36^1 is replaced by $3^{12}12^1$. Finally, 12^1 is replaced by 2^{11} resulting in $2^{11}3^{12}$.

%MktOrth Macro Options

The following options can be used with the %MktOrth macro:

Option	Description
help	(positional) "help" or "?" displays syntax summary
filter=*n*	extra filtering of the design catalog
maxn=*n*	maximum number of runs of interest
maxlev=*n*	maximum number of levels
options=lineage	construct the design lineage
options=mktex	the macro is being called from the %MktEx macro
options=mktruns	the macro is being called from the %MktRuns macro
options=parent	lists only parent designs
options=dups	suppress duplicate and inferior design filtering
options=512	adds some designs in 512 runs
outall=*SAS-data-set*	output data set with all designs
outcat=*SAS-data-set*	design catalog data set
outlev=*SAS-data-set*	output data set with the list of levels
range=*range-specification*	number of runs of interest

You can specify either of the following to display the option names and simple examples of the macro syntax:

```
%mktorth(help)
%mktorth(?)
```

filter= *n*

specifies extra design catalog filtering. Usually, you will never have to use this option. By default, `%MktOrth` filters out inferior designs. On the one hand, this process is expensive, but it also saves some time and resources by limiting the information that must be processed. Care is taken to do only the minimum amount of work to filter. However, this gets complicated. If you specify `maxlev=144`, the maximum, you will get perfect filtering of duplicates in a reasonable amount of time. `%MktOrth` uses internal and optimized constants to avoid doing extra work. Unfortunately, these constants might not be optimal for any other `maxlev=` value. This is almost never going to be a real issue. If however, you want to specify both `maxlev=` and ensure you get better filtering, specify `filter=`*n* (for some *n*) and you might get a few more inferior designs filtered out. The *n* value increases how deeply into the catalog `%MktOrth` searches for inferior designs. Larger values find more designs to exclude at a cost of greater run time. The following steps use both the default filtering and specify `filter=1000`:

```
%mktorth(maxlev=20)
%mktorth(maxlev=20, filter=1000)
```

The latter specification removes on the order of thirty more designs but at cost of much slower run time. Actually, `filter=27` is large enough for this example, and it has a negligible effect on run time, but you have to run the macro multiple times to figure that out. Values of a couple hundred or so are probably always going to be sufficient.

maxlev= *n*

specifies the maximum number of levels to consider. Specify a value n, such that $2 \leq n \leq 144$. The default is `maxlev=50`. This option controls the number of `x` variables in the `outlev=` data set. It also excludes from consideration designs with factors of more than `maxlev=` levels so it affects the number of rows in the output data sets. Note that specifying `maxlev=`*n* does not preclude designs with more than n-level factors from being used as parents for other designs, it just precludes the larger designs from being output. For example, with `maxlev=3`, the design $3^{12}12^1$ in 36 runs is used to make $2^{11}3^{12}$ before the $3^{12}12^1$ design is discarded. Specifying smaller values will make the macro run faster. With the maximum, `maxlev=144`, run time to generate the entire catalog can be on the order of several minutes.

maxn= *n*

specifies the maximum number of runs of interest. Specifying small numbers (e.g. $n \leq 200$) will make the macro run faster.

options= *options-list*

specifies binary options. By default, none of these options are specified. Specify one or more of the following values after `options=`.

`lineage`
construct the design lineage, which is the set of instructions on how the design is made.

`mktex`
specifies that the macro is being called from the `%MktEx` macro and just the `outlev=` data set is needed. The macro takes short cuts to make it run faster doing only what the `%MktEx` macro needs.

`mktruns`
specifies that the macro is being called from the `%MktRuns` macro and just the `outlev=` data set is needed. The macro takes short cuts to make it run faster doing only what `%MktRuns` needs.

`parent`
specifies that only parent designs should be listed.

`dups`
specifies that the `%MktRuns` macro should not filter out duplicate and inferior designs from the catalog. This can be useful when you are creating a data set for the `cat=` option in the `%MktEx` macro.

`512`
adds some larger designs in 512 runs with mixes of 16, 8, 4, and 2-level factors to the catalog, which gives added flexibility in 512 runs at a cost of much slower run time. This option replaces the default parent design $4^{160}32^1$ with $16^{32}32^1$.

outall= *SAS-data-set*
specifies the output data set with all designs. This data set is not created by default. This data set is like the `outlev=` data set, except larger. The `outall=` data set includes *all* of the `%MktEx` design catalog, including all of the smaller designs that can be trivially made from larger designs by dropping factors. For example, when the `outlev=` data set has `x2`=2 `x3`=2, then the `outall=` data set has that design and also `x1=2 x3=1`, `x1=1 x3=2`, and `x1=1 x2=1`. When you specify `outall=` you must also specify a reasonably small `range=` or `maxn=` value. Otherwise, the `outall=` specification will take a *long* time and create a *huge* data set, which will very likely be too large to store on your computer.

outcat= *SAS-data-set*
specifies the output data set with the catalog of designs that the `%MktEx` macro can create. The default is `outcat=MktDesCat`.

outlev= *SAS-data-set*
specifies the output data set with the list of designs and 50 (by default) more variables, `x1-x50`, which includes: `x2` – the number of two-level factors, `x3` – the number of three-level factors, and so on. The default is `outlev=MktDesLev`. The number of `x` variables is determined by the `maxlev=` option.

range= *range-specification*
specifies the number of runs of interest. Specify a range involving `n`, where `n` is the number of runs. Your range specification must be a logical expression involving `n`. Examples:
`range=n=36`

```
range=18 le n le 36
range=n eq 18 or n eq 36
```

%MktOrth Macro Notes

This macro specifies `options nonotes` throughout most of its execution. If you want to see all of the notes, submit the statement `%let mktopts = notes;` before running the macro. To see the macro version, submit the statement `%let mktopts = version;` before running the macro.

The Orthogonal Array Catalog

This section contains information about the orthogonal array catalog. While this information is interesting and sometimes useful, this section can be skipped by most readers.

The `%MktOrth` macro maintains the orthogonal array catalog that the `%MktEx` and `%MktRuns` macros use. The `%MktOrth` macro instructs the `%MktEx` macro on both what orthogonal arrays exist and how to make them. In most cases, the `%MktOrth` macro is called by the `%MktEx` and `%MktRuns` macros and you never have to worry about it. However, you can use the `%MktOrth` macro to find out what designs exist in the orthogonal array catalog. This can be particularly useful for the orthogonal array specialist who seeks to understand the catalog and find candidates for undiscovered orthogonal arrays. Most designs explicitly appear in the catalog. Others are explicitly entered into the catalog, but they normally do not appear. Still others do not explicitly appear in the catalog, but the `%MktEx` macro can still figure out how to make them. These points and other aspects of the orthogonal array catalog are discussed in this section.

First, consider a design that is explicitly in the catalog. The following steps create a list of orthogonal arrays in 18 runs, display that list, then create the design using the `%MktEx` macro:

```
%mktorth(range=n=18, options=lineage)

data _null_;
   set;
   design = compbl(design);
   put 'Design:  ' design / 'Lineage: ' lineage /;
   run;

%mktex(6 3 ** 6, n=18)
```

Of course the first two steps are not needed if your only goal is to make the design. The results are as follows:

```
Design:  2 ** 1 3 ** 7
Lineage: 18 ** 1 : 18 ** 1 > 3 ** 6 6 ** 1 : 6 ** 1 > 2 ** 1 3 ** 1

Design:  2 ** 1 9 ** 1
Lineage: 18 ** 1 : 18 ** 1 > 2 ** 1 9 ** 1 (parent)

Design:  3 ** 6 6 ** 1
Lineage: 18 ** 1 : 18 ** 1 > 3 ** 6 6 ** 1 (parent)
```

Three designs are listed. The third is the parent array $3^6 6^1$. The first is a child array $2^1 3^7$, which is constructed from the parent array $3^6 6^1$ by replacing the six-level factor with a two-level factor and a three-level factor. This can be seen in the lineage, which is explained in more detail later in this section. The second array is the full-factorial design $2^1 9^1$. All three of these arrays are explicitly in the catalog and can be directly produced by the `%MktEx` macro.

The catalog actually contains one more array that is created, but by default, it is discarded as inferior before the catalog is output or before `%MktEx` can use the catalog. You can use the `dups` option to see the duplicate and inferior arrays that would normally be discarded. The following steps illustrate this option:

```
%mktorth(range=n=18, options=lineage dups)

data _null_;
   set;
   design = compbl(design);
   put 'Design:  ' design / 'Lineage: ' lineage /;
   run;
```

The results are as follows:

```
Design:  2 ** 1 3 ** 7
Lineage: 18 ** 1 : 18 ** 1 > 3 ** 6 6 ** 1 : 6 ** 1 > 2 ** 1 3 ** 1

Design:  2 ** 1 3 ** 4
Lineage: 18 ** 1 : 18 ** 1 > 2 ** 1 9 ** 1 : 9 ** 1 > 3 ** 4

Design:  2 ** 1 9 ** 1
Lineage: 18 ** 1 : 18 ** 1 > 2 ** 1 9 ** 1 (parent)

Design:  3 ** 6 6 ** 1
Lineage: 18 ** 1 : 18 ** 1 > 3 ** 6 6 ** 1 (parent)
```

The new array is the second array in the list, $2^1 3^4$. It is made from the full-factorial design, $2^1 9^1$ by replacing the nine-level factor with 4 three-level factors. The resulting array, $2^1 3^4$, is inferior to $2^1 3^7$ in

that the former has three fewer three-level factors than the latter. Hence, by default, it is discarded. This does not imply, however, that the $2^1 3^4$ design is a simple subset of the larger design, $2^1 3^7$. The smaller design might or might not be a subset, but typically it will not be. By default, the `%MktEx` macro operates under the assumption that orthogonal and balanced factors are interchangeable, so it always prefers arrays with more factors to arrays with fewer factors. By default, it will never produce the array $2^1 3^4$ that is based on the full-factorial design, $2^1 9^1$. It will always create $2^1 3^4$ from the first five columns of $2^1 3^7$.

The `%MktEx` macro provides you with a way to get these designs that are automatically excluded from the catalog. More generally, when there are multiple designs that meet your criteria, it gives you a way to explicitly choose which design you get. All you have to do is run the `%MktOrth` macro yourself, select which design you want, and feed just that one line of the catalog into the `%MktEx` macro. The following steps illustrate this option:

```
%mktorth(range=n=18, options=lineage dups)

data lev;
   set mktdeslev;
   where lineage ? '3 ** 4';
   run;

%mktex(2 3 ** 4,                    /* 1 two-level and 4 three-level factors*/
       n=18,                        /* 18 runs                              */
       cat=lev,                     /* OA catalog comes from lev data set   */
       out=alternative)             /* name of output design                */

%mktex(2 3 ** 4, n=18, out=default)
```

The `where` clause in the DATA step selects only one of the arrays since the other arrays that have more than 4 three-level factors have '3 ** 6' not '3 ** 4' in their lineage.

The default and the alternative (or "inferior") arrays are displayed next:

Default Array					Alternative Array				
x1	x2	x3	x4	x5	x1	x2	x3	x4	x5
1	1	1	1	1	1	1	1	1	1
1	1	2	1	3	1	1	2	3	2
1	1	3	2	3	1	1	3	2	3
1	2	1	3	1	1	2	1	3	3
1	2	2	2	2	1	2	2	2	1
1	2	3	2	1	1	2	3	1	2
1	3	1	3	2	1	3	1	2	2
1	3	2	1	2	1	3	2	1	3
1	3	3	3	3	1	3	3	3	1
2	1	1	2	2	2	1	1	1	1
2	1	2	3	1	2	1	2	3	2
2	1	3	3	2	2	1	3	2	3
2	2	1	1	3	2	2	1	3	3
2	2	2	3	3	2	2	2	2	1
2	2	3	1	2	2	2	3	1	2
2	3	1	2	3	2	3	1	2	2
2	3	2	2	1	2	3	2	1	3
2	3	3	1	1	2	3	3	3	1

If only the first three factors had been requested, the two arrays would have been the same. This is because the first three factors form a full-factorial design. The remaining factors are different in the two arrays. Both are orthogonal and balanced, however, they differ in terms of their aliasing structure. In other words, they differ in terms of which effects are confounded. The GLM procedure can be used to examine the aliasing structure of a design. It provides a list of be estimable functions. Since GLM is a modeling procedure, it requires a dependent variable. Hence, the first step is to add a dependent variable y, which can be anything for our purposes, to each design. The following steps display the aliasing structure for main effects only:

```
data d;
   set default;
   y = 1;
   run;

data i;
   set alternative;
   y = 1;
   run;

proc glm data=d;
   ods select galiasing;
   model y = x1-x5 / e aliasing;
   run; quit;

proc glm data=i;
   ods select galiasing;
   model y = x1-x5 / e aliasing;
   run; quit;
```

The results for the default design are as follows:

```
General Form of Aliasing Structure

Intercept
x1
x2
x3
x4
x5
```

The results for the alternative design are as follows:

```
General Form of Aliasing Structure

Intercept
x1
x2
x3
x4
x5
```

The two results are the same, and they show that all effects can be estimated.

Next, we will try the same thing, but this time adding two-way interactions to the model. The following steps illustrate:

```
proc glm data=d;
   ods select galiasing;
   model y = x1-x5 x1|x2|x3|x4|x5@2 / e aliasing;
   run; quit;

proc glm data=i;
   model y = x1-x5 x1|x2|x3|x4|x5@2 / e aliasing;
   run; quit;
```

Note that the `x1-x5` list could be omitted from the independent variable specification, but leaving it in serves to list the main-effect terms first. The results for the default design are as follows:

General Form of Aliasing Structure

Intercept
x1
x2
x3
x4
x5
x1*x2
x1*x3
x2*x3
x1*x4
x2*x4
x3*x4
x1*x5
x2*x5
x3*x5
x4*x5

All main effects and two-factor interactions are estimable.

The results for the alternative design are as follows:

```
  General Form of Aliasing Structure

Intercept - 2*x2*x5 - 2*x3*x5 - 2*x4*x5
x1
x2 + 2*x2*x5 + 0.5*x3*x5 - 1.5*x4*x5
x3 - 1.5*x2*x5 + 2*x3*x5 + 0.5*x4*x5
x4 + 0.5*x2*x5 - 1.5*x3*x5 + 2*x4*x5
x5 + 2*x2*x5 + 2*x3*x5 + 2*x4*x5
x1*x2
x1*x3
x2*x3 + 0.5*x2*x5 - 0.5*x3*x5
x1*x4
x2*x4 - 0.5*x2*x5 + 0.5*x4*x5
x3*x4 + 0.5*x3*x5 - 0.5*x4*x5
x1*x5
```

The first estimable function is a function of the intercept and 3 two-way interactions. In other words, the intercept is confounded with 3 of the two-way interactions. We can estimate the intercept if the two-way interactions are zero or negligible. The second estimable function is the first factor, `x1`. It is not confounded with any other effect in the model. The third estimable function is a combination of the second factor, `x2` and 3 two-way interactions. In most cases, lower-order terms are confounded with higher-order terms. Both sets of results assume that all three-way and higher-way interactions are zero, since they are not specified in the model. You can add all interactions to the models as follows:

```
proc glm data=d;
   ods select galiasing;
   model y = x1-x5 x1|x2|x3|x4|x5@5 / e aliasing;
   run; quit;

proc glm data=i;
   model y = x1-x5 x1|x2|x3|x4|x5@5 / e aliasing;
   run; quit;
```

The first two terms from the default design are as follows:

```
Intercept + 27*x1*x5 + 37.75*x2*x5 + 120.5*x1*x2*x5 + 45*x1*x3*x5 +
62.917*x2*x3*x5 + 200.83*x1*x2*x3*x5 + 9*x4*x5 + 63*x1*x4*x5 + 97.083*x2*x4*x5 +
269.17*x1*x2*x4*x5 + 33*x3*x4*x5 + 129*x1*x3*x4*x5 + 202.58*x2*x3*x4*x5 +
522.17*x1*x2*x3*x4*x5

x1 - 13.5*x1*x5 - 28.75*x2*x5 - 80*x1*x2*x5 - 22.5*x1*x3*x5 - 47.917*x2*x3*x5 -
133.33*x1*x2*x3*x5 - 22.5*x1*x4*x5 - 61.083*x2*x4*x5 - 159.67*x1*x2*x4*x5 -
6*x3*x4*x5 - 43.5*x1*x3*x4*x5 - 124.58*x2*x3*x4*x5 - 307.67*x1*x2*x3*x4*x5
```

Again, lower-order terms are confounded with higher-order terms. This is the nature of using anything less than a full-factorial design. You typically assume that higher-order interactions are zero or

negligible and hope for the best.

At least in terms of the aliasing structure and the two-way interactions, it appears that the default design is better than the alternative design. There are no guarantees, and if the aliasing structure is ever really important to you, you should consider and evaluate the other arrays in the catalog.

The %MktOrth macro maintains a very large design catalog. However, the number of possible orthogonal arrays is infinite, and even for finite n, the total number starts getting very large after $n = 143$. Hence, it is impossible for the catalog to be complete. Still, there are some orthogonal designs are not in the %MktOrth catalog that the %MktEx macro still knows how to make. They are in some sense larger than the arrays that exist in the catalog, but they have a regular enough structure that they can be created without an explicit lineage from the catalog. This is illustrated in the following steps:

```
%mktorth(range=n=12*13, options=lineage)

proc print; run;

%mktex(12 13, n=12*13)
```

The catalog has only one design in $12 \times 13 = 156$ runs, and it is a Hadamard matrix with 155 two-level factors. The catalog entry is as follows:

Obs	n	Design	Reference	Lineage
1	156	2 **155	Hadamard	2 ** 155 (parent)

Still, the %MktEx macro can construct a full-factorial design with a twelve-level and thirteen-level factor. The following steps show the part of the catalog that contains full-factorial designs:

```
%mktorth;

proc print; where reference ? 'Full'; run;
```

The largest n in the results (not shown) is 143. The goal is for the catalog to have complete coverage up to 143 runs and good coverage beyond that. Full-factorial designs beyond 143 runs are not needed for making child arrays, so they are not included. However, %MktEx is capable of recognizing and making many full-factorial designs with more than 143 runs. The following steps provide another example:

```
%mktorth(range=n=1008, options=lineage)

%mktex(2 ** 1007, n=1008)
```

There are no designs with 1008 runs in the catalog, but the %MktEx macro recognizes that this as a Hadamard design from the Paley 1 family, and makes it using the same code that makes smaller Hadamard matrices. (More precisely, it recognizes 1008 as 4×252, and it recognizes $252 - 1$ as prime and makable with the Paley 1 construction. The final design is the Kronecker product of Hadamard matrices of order 4 and of order 252.)

There is one more class of designs that the %MktEx macro can find even when they are not in the catalog. In the following example, the %MktEx macro finds the design 29^{30} in 841 runs:

```
%mktorth(range=n=29 * 29, options=lineage)

%mktex(29 ** 30, n=29*29)
```

This is a regular fractional-factorial design (the number of runs and all factor levels are a power of the same prime, 29) that PROC FACTEX (which the `%MktEx` macro calls) can find even though it is not in the `%MktOrth` catalog.

Similarly, in 256 runs, there are designs where the factor levels are powers of two that are not in the catalog, yet the `%MktEx` macro can make them. This is illustrated in the following steps:

```
%mktorth(range=n=256, options=lineage, maxlev=64)

proc print data=mktdeslev; where x64 and x4 ge 10; run;

%mktex(4 ** 10 64, n=256)
```

The results of the PROC PRINT step (not shown) show that the design $4^{10}64^1$ is not in the design catalog. Nevertheless, the tools in the `%MktEx` macro (specifically, PROC FACTEX) succeed in constructing this design. (Note that mixes of 2, 4, 8, and 16 level-factors in 256 runs are completely covered by the catalog.)

In 128 runs, most designs are created directly by the `%MktEx` macro using information in the design lineage. However, some, even though they are explicitly in the catalog, are created by PROC FACTEX without using the design lineage from the catalog.

Next, we will return to the topic of design lineage. The design lineage is a set of instructions for making a design with smaller level factors from a design with higher-level factors. Previously, we saw the following design in 18 runs:

```
Design:  2 ** 1 3 ** 7
Lineage: 18 ** 1 : 18 ** 1 > 3 ** 6 6 ** 1 : 6 ** 1 > 2 ** 1 3 ** 1
```

It starts in the `%MktEx` macro as a single 18-level factor. That is what the first part of the lineage, '`18 ** 1`' specifies. Then the 18-level factor is replaced by 6 three-level factors and one six-level factors. This is specified by the lineage fragment: `18 ** 1 > 3 ** 6 6 ** 1`. Finally, the six-level factor is replaced by a two-level factor and a three level factor: `6 ** 1 > 2 ** 1 3 ** 1`. The code that makes $3^6 6^1$ in 18 runs is the same code that replaces an 18-level factor in the design $3^{18}18^1$ in 54 runs, or replaces an 18-level factor in 72, 108, or more runs.

The following steps show the lineage for the design $2^7 4^5 8^{15}$ in 128 runs:

```
%mktorth(range=n=128, options=lineage)

data _null_;
   set mktdeslev;
   if x2 eq 7 and x4 eq 5 and x8 eq 15;
   design = compbl(design);
   put design / lineage /;
   run;
```

The design is shown on the first line of the following output and the lineage on the second:

```
2 ** 7 4 ** 5 8 ** 15
8 ** 16 16 ** 1 : 16 ** 1 > 4 ** 5 : 8 ** 1 > 2 ** 4 4 ** 1 : 4 ** 1 > 2 ** 3
```

The parent array is $8^{16}16^1$ not a single 128-level factor. The starting point is a single n-level factor for all arrays under 145 runs with the exception of some in 128 runs. Then the sixteen-level factor is replaced by 5 four-level factors. One of the eight-level factors is replaced by $2^4 4^1$. A four-level factor is replaced by 3 two-level factors.

The following steps show a lineage for the design 2^{31} in 32 runs:

```
%mktorth(range=n=32, options=lineage)

data _null_;
   set mktdeslev;
   if x2 eq 31;
   design = compbl(design);
   put design / lineage /;
   run;
```

The design is shown on the first line of the following output and the lineage on the second:

```
2 ** 31
32 ** 1 : 32 ** 1 > 4 ** 8 8 ** 1 : 8 ** 1 > 2 ** 4 4 ** 1 : 4 ** 1 > 2 ** 3
```

This shows that the design starts as a single 32-level factor. It is replaced by $4^8 8^1$. Next, 8^1 is replaced by $2^4 4^1$. Finally, 4^1 is replaced by 2^3. Implicit in these instructions is the fact that substitutions can occur more than once. In this case, every four-level factor is replaced by 3 two-level factors. While multiple substitutions can occur for higher-level factors, in practice, multiple substitutions usually occur for only four-level factors and sometimes six-level or eight-level factors.