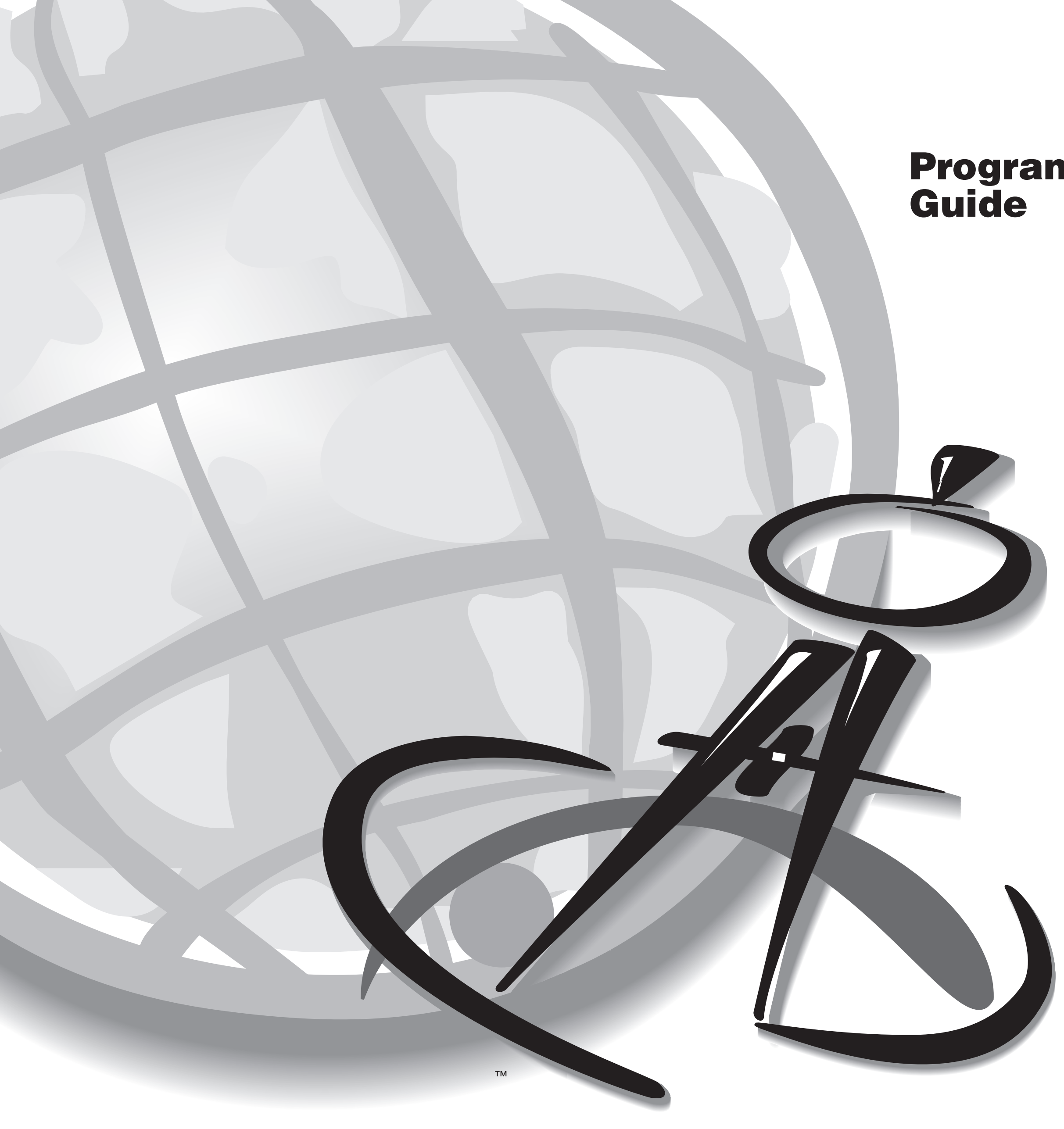
Programmer's Guide



ArcCAD®

The GIS for AutoCAD®

Contents

Introduction

The real power of either AutoCAD® or ArcCAD® lies in the way you customize your software to meet the specific requirements of your work. AutoCAD provides you with the tools in AutoLISP® and AutoCAD Development System™ (ADS) to let you write your own programs or customize virtually every aspect of your software. Over the years, these powerful AutoCAD tools have given third-party application builders a tremendous developmental environment within which to write a wide variety of vertical applications on top of the generic tools provided in the general purpose AutoCAD design package.

Along the same lines, the ArcCAD AutoLISP and ADS extensions described in this guide let you customize the software and assist you in building custom applications on top of ArcCAD. The set of AutoLISP and ADS functions described in this guide is a small subset of functions specifically designed to access and manipulate the ArcCAD geographic information system (GIS) data sets. Since ArcCAD runs within AutoCAD, and ArcCAD features are represented by AutoCAD entities, for the most part, you must use this guide in association with the *AutoCAD Customization Guide* supplied with your AutoCAD software.

The most important concept of the ArcCAD data model is the relationship between AutoCAD entities and ArcCAD geographic features. These relationships are intelligently linked to enable communication between AutoCAD entities and ArcCAD spatial features. For the most part, an AutoCAD entity behaves identically to an ArcCAD feature. For example, an entity may be as simple as a single line segment or as complicated as a nested block. The same is true in ArcCAD. A feature may be as simple as an arc (linear) feature or as complicated as a polygon feature comprising a set of linear features, making up the polygon boundary, and one or more label points.

In AutoCAD, every entity in a drawing has a unique *entity name*. Using a variety of AutoLISP and ADS functions, you can access the data associated with that entity. You can group a set of entities and store them in *selection sets* for further use in a given AutoCAD session. A similar approach is used in ArcCAD to access GIS data sets. Every feature in the data set has a unique *feature name*. You can group a set of features and store them in *feature selection sets*. The intelligent link between the entities and features gives you enormous power. For example, given an entity name, you can access its corresponding feature in your GIS data set and vice versa. These kinds of tools are described in this guide to assist you in customizing the ArcCAD software and give you the ability to write a variety of GIS applications in AutoLISP and ADS.

What is AutoLISP?

AutoLISP is an implementation of the LISP programming language embedded within the AutoCAD package. AutoLISP lets users and AutoCAD developers write macro programs and functions in a powerful high-level language that is well suited to graphics applications. LISP is easy to learn and use, and is very flexible (*AutoLISP Programmer's Reference, Release 11*, Autodesk®, Inc.).

The AutoLISP programming language is supplied with each copy of AutoCAD. Refer to Part 2 of the *AutoCAD Customization Guide* for complete information about AutoLISP. The ArcCAD extension to AutoLISP is a small subset of AutoLISP functions specifically designed to access the ArcCAD GIS data sets. These functions are supplied with every copy of the ArcCAD software. There is no special configuration or installation procedure required to access these functions in ArcCAD as long as you have ArcCAD up and running.

What is ADS?

ADS is a C-language programming interface embedded in AutoCAD software for developing C-based applications. The ArcCAD ADS tools described in this guide are a set of external functions loaded by, and called from, the AutoLISP interpreter and extend the power of ADS to enable the developer to create GIS applications. Although the programming environment and programming interface differ from AutoLISP, for the most part, ADS functions described in this guide are functionally equivalent to AutoLISP functions. All the ADS functions described in this guide have the same name as the comparable AutoLISP functions, except for the prefix **arc_**. This similarity between ADS functions and AutoLISP functions makes it relatively easy to convert programs from AutoLISP to C.

AutoLISP versus ADS

Although for the most part AutoLISP and ADS tools are functionally equivalent, the choice of whether to use AutoLISP or ADS is up to the developer. ADS applications tend to be more efficient in terms of speed and memory usage; they can also directly access some facilities that AutoLISP cannot, such as the host operating system and hardware. This makes them well suited for applications that require considerable computation or interaction with the host environment. On the other hand, they can be more time consuming and expensive to develop and maintain. AutoLISP is better suited for applications where maintenance and development costs are more important considerations than performance; this is often the case for highly interactive applications that are bound by user response time (*ADS Programmer's Reference Release 11,* Autodesk, Inc.). Since more ArcCAD functions are available to the AutoLISP programmer, we encourage you to use AutoLISP.

Organization of ArcCAD databases

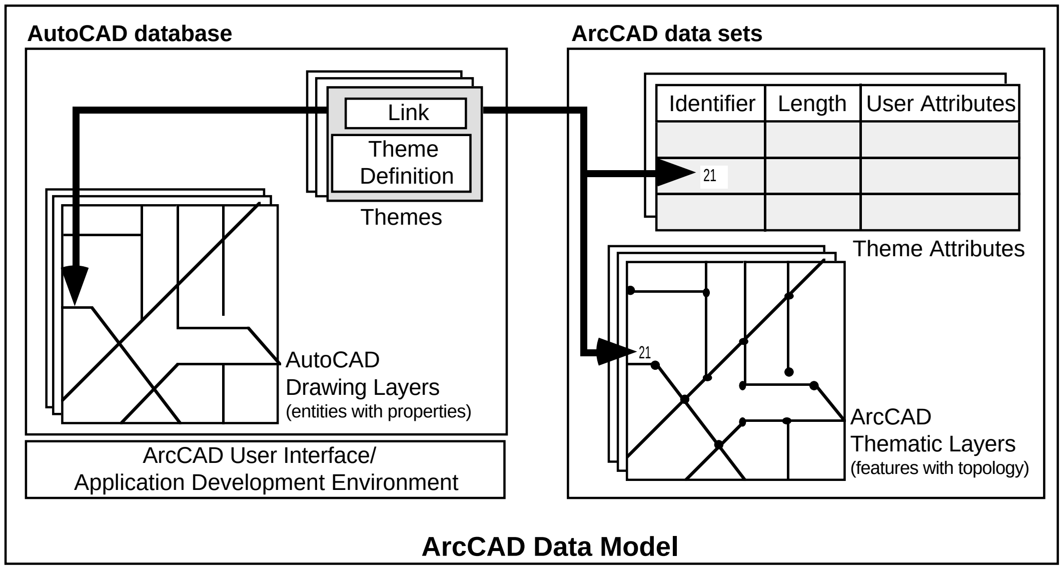
ArcCAD provides a geographically referenced spatial database on which GIS operations may be performed. Links to the AutoCAD database are provided to manage the relationship between the ArcCAD GIS data set and the AutoCAD CAD database. In ArcCAD, the CAD database is used to store graphical representations (drawings) of the geographically referenced spatial data sets (coverages) stored as ArcCAD databases. ArcCAD also provides a set of GIS operators that can be used to automate, manipulate, analyze and display the spatial GIS data set to analyze complex spatial relationships.

The ArcCAD data model consists of a CAD database and a GIS data set together with information that intelligently links the two. The most important concept of the ArcCAD data model is this relationship between the AutoCAD entities and the geographically referenced objects that represent geographic features stored as ArcCAD databases.

The relationship between the CAD and GIS data models is represented by a *theme*. The theme defines the relationship between AutoCAD drawing entities and GIS features stored in the GIS data set.

A GIS data set is a generic term used throughout this guide. A GIS data set may consist of PC ARC/INFO-compatible coverages and database data files.

Here is the conceptual organization of the ArcCAD data model:


AutoCAD database
ArcCAD data sets
Link
Theme Definition
Themes
Identifier
Length
User Attributes
21
Theme Attributes
AutoCAD Drawing Layers (entities with properties)
ArcCAD Thematic Layers (features with topology)
ArcCAD User Interface/ Application Development Environment
ArcCAD Data Model

What is a coverage?

Under the DOS and MS Windows operating systems, coverages are implemented as subdirectories having the same name as the coverage under which all coverage data files reside. The directory structure of an ArcCAD coverage is outlined below with a brief description of the most common coverage files:

AAT.DBF	- Arc attribute table
ARC	- Arc coordinates and topology
ARF	- Arc cross-reference file
BND.DBF	- Coverage minimum and maximum coordinates
CNT	- Polygon centroid table
LAB	- Label point coordinates and topology
LOG	- Coverage or workspace history file
MSK	- Edit area masks
PAL	- Polygon topology
PAT.DBF	- Polygon or point attribute table
PFF	- Polygon filter file
PRF	- Polygon or point cross-reference file
TIC.DBF	- Tic coordinates and IDs
TOL	- Coverage processing tolerances
TXT	- Coverage annotation features

Some of the above files contain locational information, while others contain corresponding attribute information, with pointers that link the two, maintained in other cross-reference files. The following table explains the relationship between some of the files that store locational information and their implied attribute tables.

	Location	**Attributes**
TIC	TIC.DBF	NONE
ARC	ARC	AAT
POLYGON	PAL,CNT	PAT
LABEL	LAB	PAT
ANNOTATION	TXT	NONE

It is not necessary to understand the details of file structure, but you should be aware of the relationships between individual files in a coverage. The tools described in this guide automatically maintain the information within these files for you. It is important that the developer not alter the contents of these files without using the tools provided in this guide. For example, if you alter the geometry of a feature (locational information in an ARC file), you must rebuild the topology using the **build** or **clean** operations to reflect these changes in other files and maintain the integrity of the database.

What is topology?

An ArcCAD coverage explicitly represents all geographic features by sets of lines (also referred to as arcs) and points, and as relationships between connected lines and points. For example, an area or polygon is defined by the set of arcs that makes up its boundary, where an arc is the border between two polygons. Also, an arc could be part of a path connecting other arcs. For example, arcs can be used to represent streets and the routes that pass through them.

The relationships used to represent the connectivity or contiguity of these features are referred to as topology. Topology is the highest level of generalization at which geographic features can be stored. By storing information about the location of a feature relative to other features, topology provides the basis for many kinds of geographic analysis without having to access the absolute locations held in the coordinate files (e.g., connectivity, route finding and contiguity are all derived through topology).

For a detailed explanation of topological relationships on individual feature classes and how these relationships are maintained, refer to the ‘ArcCAD concepts’ chapter of the *ArcCAD User’s Guide.* It is important that you understand the concept of topological relationships before you start building applications using the tools described in this guide.

Before you start

Before you start using this guide, you should understand the organization of ArcCAD databases, how themes are stored, and the relationships between AutoCAD entities and ArcCAD spatial features. You should also be familiar with topological relationships between features, the coverage data model and how coverages are stored in ArcCAD. If you are not yet acquainted with the ArcCAD data model, you should read the concepts chapter in the *ArcCAD User’s Guide*.

We also assume that you are a reasonably proficient AutoCAD and ArcCAD user; that is, you know the AutoCAD and ArcCAD commands and the general concepts of AutoCAD. We also assume that you have some experience in programming in either AutoLISP or ADS, or both. If you are not familiar with any one of these, we strongly suggest you consult the appropriate AutoCAD programmer reference manuals to familiarize yourself with AutoLISP or ADS. This is required because the tools described in this guide are a small subset of extensions to the AutoLISP and ADS functions provided by Autodesk, Inc.

Organization of the guide

As mentioned earlier, the *ArcCAD Programmer's Guide* will assist you in customizing the software or building specific GIS applications using the ArcCAD AutoLISP and ADS tools. Although the functional tools provided in both AutoLISP and ADS are identical in nature, the programming environment and programming interface for AutoLISP and ADS are entirely different. In order to avoid confusion, this guide is divided into two major sections:

Section 1: ArcCAD Extensions to AutoLISP
Section 2: ArcCAD Extensions to ADS

Each section is self-contained with its own table of contents and index. We hope this organization will assist you in using this guide.

ArcCAD limitations

The following limitations apply to certain ArcCAD functions and the ArcCAD data model in general. These limitations may affect your ArcCAD applications. Keeping these limitations in mind may save you time and effort while designing specific GIS applications.

Feature limitations

■ Maximum number of features in a coverage	262,144
■ Maximum number of *tic* features in a coverage	5,000
■ Maximum number of *arcs* (linear features) per polygon	10,000
■ Maximum number of *label points* per polygon	100
■ Maximum number of *vertices* per arc	500
■ Maximum number of concurrent *feature selection sets*	128

Other general limitations

Coordinate rounding

Coverages containing very large coordinate values may experience some small coordinate rounding during overlay operations due to single-precision data limits. This is especially true for those coverages also having a small range of x or y coordinate values. This limit may be avoided by using the **xyshift** command to apply a constant offset to the coordinates.

Themes

- The maximum length of a theme name is 31 characters.
- The maximum number of themes that can be defined and linked to GIS data sets in a single AutoCAD drawing is 511.
- You cannot define more than one theme with the same feature class and GIS data set, with the exception of *record* themes.

GIS data sets

- The maximum length of the name of a GIS data set is 8 characters.
- The maximum length of the full pathname to a GIS data set is 64 characters. This includes the GIS data set name.
- A coverage GIS data set cannot contain both point and polygon features.

Item names

- The maximum length of an item name is 10 characters.
- The only legal characters in item names are alphabetic characters, numbers and the underscore character.
- Item names cannot begin with a number.

ArcCAD® Extensions to AutoLISP™

Contents

Chapter 1

Theme access

A theme is a collection of geographic phenomena or an organizing principle that is used to link AutoCAD® entities to geographic information system (GIS) data sets. Each theme has a unique name, a feature class, a pointer referencing a corresponding GIS data set (a PC ARC/INFO-compatible coverage or a database data file), and a symbol number. Themes are stored as point entities in the current drawing and maintained on a special layer named ESRI_THEMES. The parameters of the theme definition are maintained in the extended entity data of the point entities that are created on this special layer. This layer is not visible to the user and should always be in a *frozen* state. You must never attempt to edit the contents of this layer. Doing so can destroy the theme definitions and therefore corrupt all the links that are maintained between AutoCAD entities and ArcCAD® features.

The theme manipulation functions described in the following section maintain and manipulate the extended entity data and the links for you. These functions are used to manage themes in your drawing and maintain the links between the ArcCAD features and the corresponding AutoCAD entities. For example, using the **thmdef** function, you can define a theme to create or display GIS data sets. You can use the **thmdel** function to delete a theme and the links to the GIS data set. Similarly, using the **thmexi** function, you can check the validity of a theme before performing any operation on the theme. The **thmlst** function lists the available themes in your drawing, and the **thmget** function lets you retrieve the contents of the theme definition in the form of a list. The list obtained in this fashion can be used to modify the definition of the theme using the **thmmod** function.

The following are some of the limitations of the theme access functions:

- The maximum length of a theme name is 31 characters.

- The maximum number of themes that can be defined and linked to GIS data sets in a single AutoCAD drawing is 511.

- You cannot define more than one theme with the same feature class and GIS data set, with the exception of *record* themes.

Theme manipulation functions

The following functions can be used to create, modify, list and delete some of the parameters that define a theme.

(thmdef *list*)

This function creates a new theme in the drawing. If *theme* is successfully created, the name of the theme is returned. If *theme* cannot be created, `nil` is returned. There are eight possible feature classes available to the user: Annotation, Image, Line, Point, Polygon, Record (dBASE), Record (SQL) and Tic.

The format of the input list is as follows:

```
(    (-1 . 0)             dummy ename
     (1 . themename)      theme name
     (2 . class)          feature class
     (3 . GIS data set)   GIS data set name
     (57 . increment)     User-ID increment
     (58 . next id)       next available User-ID
     (60 . symbol)        symbol number between 0 and 100
     (62 . visibility)    theme visibility
)
```

When defining an SQL Record theme, the *list* argument has the following format:

```
(    (-1 . 0)
     (1 . themename)
     (2 . class)
     (5 . user name)      SQL connection user name
     (6 . password)       SQL connection password
     (7 . query1)         1st 128 characters of SQL query
                          expression
     (8 . query2)         remaining characters of SQL query
     (9 . connection)     SQL connection name
)
```

This function does not perform any user interaction. All of the parameters must be supplied. If user interaction is desired, use the **cmd** function to invoke the user command that defines themes.

For example, to create a line theme named ‘SAMPLE’ with a GIS data set of ‘C:\DEMO\SAMPLE’ that uses the default line symbol, the following code fragment could be used.

```
(setq theme
   (thmdef
      (list
         (cons -1 0)
         (cons 1 “sample”)
         (cons 2 “line”)
         (cons 3 “c:\\demo\\sample”)
         (cons 60 0)
      )
   )
)
(if (= theme “sample”)
   (princ “theme defined\n”)
   (princ “theme not defined\n”)
)
```

The next example shows how to create an SQL theme with the name `‘SQLSAMPLE’`, connection name of `‘DB3’` (dBASE III), user name of `‘JSmith’` and password of `‘joshua’`. The query string is an SQL expression that identifies the SQL catalog, schema and table, respectively, as `DATA.WATER.PONDS`.

```
(setq theme
   (thmdef
      (list
         (cons -1 0)
         (cons 1 “sqlsample”)
         (cons 2 “sql")
         (cons 5 “jsmith”)
         (cons 6 “joshua”)
         (cons 7 “select * from data.water.ponds”)
         (cons 9 “DB3”)
      )
   )
)
(if (= theme “sqlsample”)
   (princ “theme defined\n”)
   (princ “theme not defined\n”)
)
```

Note: If it happens to be the first theme created in your drawing, the software first creates a frozen, invisible layer called ESRI_THEMES.

(thmdel *theme option*)

This function removes a theme from the drawing. The actual output of this function is dependent upon the value of *option* provided. This function will either return a theme name or `nil`.

If *option* evaluates to 0, only the links associated with *theme* will be removed from the drawing. *Theme* will remain defined in the drawing. If the links are successfully removed, the name of the theme will be returned. If the links cannot be removed, `nil` will be returned.

If *option* evaluates to 1, the associated links *and* the theme definition will be removed from the drawing. If *theme* is successfully deleted, the name of the theme will be returned. If *theme* cannot be deleted, `nil` will be returned.

When a theme is deleted from the drawing, the associated links are also removed from all entities belonging to the theme. In addition, all feature selection sets derived from the theme are removed.

For example, consider the following code fragment:

```
; Remove only the links of theme named ‘SAMPLE’
(setq theme (thmdel “sample” 0))
(if (= theme “sample”)
     (princ “removed links of theme\n”)
     (princ “links were not removed\n”)
)
;
; Remove definition & links for theme named ‘ROADS’
(setq theme (thmdel “roads” 1))
(if (= theme “roads”)
     (princ “theme and links deleted\n”)
     (princ “links and theme were not removed\n”)
)
```

Note: If you delete a GIS data set referenced by a theme, it is always a good idea to use the **thmdel** function to remove the links associated with that theme. The ArcCAD command **kill** handles this automatically.

(thmexi *theme*)

This function checks for the existence of a theme in the drawing. If *theme* exists, the name of the theme is returned. If *theme* does not exist, `nil` is returned.

For example, to check for the existence of a theme named ‘SAMPLE’, the following code fragment could be used.

```
(setq theme (thmexi “sample”))
(if (= theme “sample”)
   (princ “found it\n”)
   (princ “does not exist\n”)
)
```

(thmget *theme*)

This function returns the definition of an existing theme. If *theme* exists, a list is returned that contains the theme’s name, class, GIS data set, and symbol number. If *theme* does not exist, `nil` is returned.

The list that is returned will be in a format that can be easily manipulated using the **assoc** function. The format of the returned list is

```
(     (-1 . 0)               dummy ename
      (1 . themename)        theme name
      (2 . class)            feature class
      (3 . GIS data set)     GIS data set name
      (57 . increment)       User-ID increment
      (58 . next id)         next available User-ID
      (60 . symbol)          symbol number between 0 and 100
      (62 . visibility)      theme visibility
      (63 . has data)        features present in theme
)
```

For example, to retrieve the definition of a theme named ‘ROADS’, the following code fragment could be used.

```
(setq themelist (thmget “roads”))
(if (null themelist)
   (princ “theme not defined\n”)
   (progn
      (setq name (assoc 1 themelist))
      (setq class (assoc 2 themelist))
      (setq data (assoc 3 themelist))
      (setq sym (assoc 60 themelist))
      (princ “theme defined\n”)
   )
)
```

The following additional codes are returned when *theme* is an SQL RECORD theme:

`(7 . query1)`	*1st 128 characters of SQL query expression*
`(8 . query2)`	*remaining characters of SQL query*
`(9 . connection)`	*SQL connection name*

(thmlst *[class dataset]*)

This function returns a list of the themes defined in the current drawing. If the drawing does not contain any themes, `nil` is returned. The optional *class* and *dataset* arguments return themes of specific types. Note that you cannot omit only one of the arguments. You must either omit both the *class* and *dataset* arguments or supply both of them as shown below:

```
(thmlst)
```

or

```
(thmlst class dataset)
```

The optional *class* argument determines the feature class of themes returned. The *class* argument is an integer (bit-coded) with values as shown below:

Bit value	Meaning
1	List *line* themes
2	List *polygon* themes
4	List *point* themes
8	List *tic* themes
16	List *annotation* themes
32	List *dBASE record* themes
64	List *SQL record* themes
128	List *image* themes

The bit values can be added in any combination to return a variety of theme feature classes. For example, if you wish to list all *line* and *polygon* themes, set the value of *class* to 3.

Caution: Future releases of ArcCAD might use additional **thmlst** control bits, so avoid setting bits in your applications that aren't shown in the above table. The optional *dataset* argument instructs **thmlst** to return themes based on the existence of their GIS data sets. You can retrieve themes for which a data set exists, or for which a data set does not exist, or you can ignore the existence of the data set altogether. The *dataset* argument is an integer with values as shown below:

Value	**Meaning**
0	Ignore GIS data set existence
1	GIS data set must exist
2	GIS data set must not exist

Data set existence rules

The GIS data set exists when the following conditions are met for *line*, *point*, *polygon*, *tic* and *annotation* themes:

■ The GIS data set (coverage) has been created by adding one or more features to the theme.

■ A theme is defined with its GIS data set referencing an existing coverage in the specified subdirectory.

■ For a *record* theme, the GIS data set exists if it contains at least one item definition. It is not dependent on the number of records in the file.

The following code fragments show some examples using the **thmlst** function with a combination of *class* and *dataset* arguments. These examples assume that the following themes exist in your current drawing.

Theme name	GIS data set	Feature class	Symbol
LANDUSE	*Present*POLYgon	0	
ZONING	*Not present*	POLYgon	0
ROADS	*Present*Line	0	
STREAMS	*Present*Line	0	
RIVERS	*Not present*	Line	0
WELLS	*Present*POint	0	

Example 1

List all the themes, irrespective of GIS data set existence.

The following code fragment

```
(thmlst)
```

would return the following list:

```
("LANDUSE" "ZONING" "ROADS" "STREAMS" "RIVERS" "WELLS")
```

Example 2

List only *line* and *polygon* themes and ignore GIS data set existence.

The following code fragment

```
(thmlst (+ 1 2) 0)
```

would return the following list:

```
("LANDUSE" "ZONING" "ROADS" "STREAMS" "RIVERS")
```

Example 3

List only *line* themes for which a GIS data set exists.

The following code fragment

```
(thmlst 1 1)
```

would return the following list:

```
("ROADS" "STREAMS")
```

Please note that although RIVERS is a *line* theme, it is not in the returned list because the *dataset* argument was 1 (GIS data set must exist).

(thmmod *theme list*)

This function modifies the definition of an existing theme. If *theme* is successfully modified, the name of the theme contained in *list* is returned. If the theme's name is being modified, the new name is returned. If the theme's name is not being modified, the original name is returned.

If *theme* cannot be modified, `nil` is returned. The *list* argument must be in the same format as that received from the **thmget** function. *Please note that the feature class (GIS data set) has data, SQL query and SQL connection name values that are read-only.*

For example, to change the name of a previously defined theme 'SAMPLE' to 'ROADS', the following code fragment could be used. This example assumes that the theme 'ROADS' does not exist in the current drawing.

```
(setq themelist (thmget "sample"))
(setq newlist
   (subst
      (cons 1 "roads")
      (assoc 1 themelist)
      themelist
   )
)
(setq theme (thmmod "sample" newlist))
(if (= theme "roads")
   (princ "modification complete\n")
   (princ "modification failed\n")
)
```

Chapter 2

Feature access

A comprehensive set of AutoLISP® and ADS™ functions, provided by Autodesk®, Inc., allows you to access AutoCAD® entities. In parallel to these functions, ArcCAD® supports similar functions that access features in ArcCAD databases. Using these functions, you can select features, retrieve their values and modify them. You can use AutoLISP variables to hold selection sets derived in this fashion so that you can manipulate selection sets of features.

Feature selection set manipulation

In order to manipulate features, the user must be able to indicate which objects are to be considered. This process involves the selection of features based upon combinations of spatial, graphical and attribute criteria specified as a series of logical expressions. This series of functions allows the user to perform and maintain selection sets and retrieve information concerning these objects of interest.

There are two special ArcCAD data types that are implemented to provide access to ArcCAD features: a *feature name* and a *feature selection set*. A feature name is a pointer into the ArcCAD database from which AutoLISP can find the appropriate feature and its corresponding attributes (if any exist). A feature selection set is simply a collection of feature names.

Feature selection sets

A feature selection set in ArcCAD is a set of features that are grouped together based on a series of spatial, logical and arithmetic expressions. At the command level, there is one feature selection set maintained for each theme (except *annotation*) defined in the drawing. Initially, all features belonging to a database are selected for that theme's selection set. Using the **reselect**, **aselect**, **nselect** and **clearsel** commands, the user can manipulate the feature selection sets to display desired features and their corresponding attribute values (if any exist). The last operation on a theme's feature selection set is always stored as the current feature selection set for that theme.

At the AutoLISP and ADS level, the user can create and manipulate feature selection sets using a set of feature selection functions (explained in later sections). By using these functions, the user can achieve the same results as one who uses ArcCAD commands for query and display purposes. There are, however, obvious advantages to using these functions to customize the software at the AutoLISP and ADS level.

As explained earlier, at the command level, there is always one and only one feature selection set maintained for any given theme. By using feature selection functions, the user can store multiple feature selection sets for a given theme and use any one of these feature selection sets to replace the current feature selection set for that same theme.

While making multiple feature selection sets using AutoLISP and ADS functions, the user has options to select features from the current feature selection set or from the entire GIS data set associated with that theme. For example, if the user creates a new feature selection set from the *current* feature selection set, the function behaves exactly like a **reselect** command. On the other hand, if the user specifies the *all* option while deriving a feature selection set (refer to the **fssget** function), the user has the ability to select a subset of features from the entire GIS data set of a given theme.

The functions **fssand**, **fssor** and **fssxor** are also available to the user to create new feature selection sets by combining two valid feature selection sets based on Boolean AND, OR, XOR combinations.

Features may be selected spatially or by using a series of logical expressions based on valid feature attributes in the corresponding feature attribute table for a given theme, as explained below.

Spatial selection

Spatial selection sets can be performed using the following modes:

- Circle crossing
- Circle within
- Window crossing
- Window within
- Polygon crossing
- Polygon within
- Previous (latest feature selection set)
- Interactive (by specifying the appropriate mode)

For further details on spatial selection, refer to the **fssget** function.

Logical expressions

Logical expressions in ArcCAD have three components: operands, logical operators and logical connectors.

Logical operands

- The name of an item in a data file (e.g., `STREAMS_ID`)
- A constant numerical value (e.g., `10`)
- A character string in single quotation marks (e.g., `'HIGH'`)
- An internal variable (e.g., `$RECNO`)

Logical operators

`EQ` or =	Operand-1 is equal to Operand-2
`NE` or <>	Operand-1 is not equal to Operand-2.
`GE` or >=	Operand-1 is greater than or equal to Operand-2.
`LE` or <=	Operand-1 is less than or equal to Operand-2.
`GT` or >	Operand-1 is greater than Operand-2.
`LT` or <	Operand-1 is less than Operand-2.
`CN`	Operand-1 contains the character expression Operand-2. Used with character operands only (e.g., `NAME CN 'MAIN'`).
`NC`	Operand-1 does not contain the character expression Operand-2. Used with character operands only (e.g., `NAME NC 'MAIN'`).
`IN`	Operand-1 is contained in the set of numeric constants of character strings specified in Operand-2. This set of constants or character strings must be enclosed in { } braces. The elements in the set must be separated by commas, unless they are being used to express a range, in which case, -> is used to separate the elements forming the lower- and upper-inclusive limits of the range. A range defined between two character strings is based on the ASCII number sequence, which is alphabetical. No blank spaces should separate any of the elements within the brackets.

Note: Computer roundoff can alter the values of real numbers. This can cause a problem when specifying real numbers in a logical expression that require equality. When using expressions of equality, the operands must match exactly for the match to be found. For example, the value .01139 does not equal .0114. In such cases, use an expression that includes a range of real values (i.e., “HEIGHT GT .01139 AND HEIGHT LT .01141”).

Logical connectors

`AND` For the whole expression to be evaluated as true, the logical expressions on both sides of the `AND` must be true.

`OR` For the whole expression to be evaluated as true, the logical expression on one or the other side of the `OR` must be true. The whole expression will also be evaluated as true if both logical expressions are true.

`XOR` For the whole expression to be evaluated as true, the logical expression on one and only one side of the `XOR` must be true. If both logical expressions are true or both are false, the condition will be evaluated as false.

The simplest logical expressions take the following form:

[operand-1] [logical-operator] [operand-2]

For example, the following string can be used as a simple logical expression:

```
"CLASS LT 8"
```

Up to eight logical expressions of this simple form can be combined to form more complex expressions by using the logical connectors. For example, the following string can be used as a complex logical expression:

```
"CLASS GE 2 AND CLASS LT 8 OR SUIT = 'MODERATE'"
```

Note: If the item value is of type character, the value must be enclosed in single quotes (example: SUIT = 'MODERATE') and is case sensitive.

There is no limit to the number of [operand-1] [logical-operator] [operand-2] combinations and logical connectors that can be used in a single expression. However, the entire expression must be less than 254 characters in length.

All logical operators and connectors have equal precedence. The expression is evaluated from left to right. However, parentheses can be used to change the default order of evaluation such that expressions within parentheses are performed first. Operations inside the innermost set of parentheses have the highest precedence.

Each element of a logical expression (i.e., operand, logical operator, logical connector, parenthesis) must be separated by blanks, except when using the IN operator.

Arithmetic expressions

Arithmetic expressions in ArcCAD have the following components:

Numeric operands

- An item name
- A constant (e.g., `10`)
- An internal variable (e.g., `$RECNO`)

Arithmetic operators

+	Addition
-	Subtraction
/	Division
*****	Multiplication
******	Exponentiation
LN	Logarithm Calculates the natural logarithm of the operand it precedes. The operand must be a positive number.
WD	Width computation Calculates the width in characters of the operand it precedes excluding trailing blanks. The operand must be a character item or a literal string.

Arithmetic operators have the following precedence from highest to lowest:

1) LN, WD
2) **
3) *, /
4) +, -

Operands of equal precedence are performed in sequence from left to right through the expression. Parentheses can be used to override inherent precedence. Operations within the innermost set of parentheses are performed first.

Note: There is no unary minus operator for negating an operand in ArcCAD. For example, the expression -AGE evokes an error message (instead, specify -1 * AGE). Also, all arithmetic operations in ArcCAD are performed in double precision. As a result, an expression involving integer operands may be evaluated as having a fractional part.

Examples of arithmetic expressions:

```
SUIT = ( SOIL + 2 * TERRAIN ) / 12

LAB_WIDTH = ( WD ( LABEL ) + 4 ) * 0.22
```

Display width limitations

A number assigned to a numeric type item with 0 decimal places that exceeds the defined width for its item definition is displayed as asterisks (***). The item value is lost.

The values of a numeric type item defined with 1 or more decimal places that exceed the defined width for that item will be displayed in scientific notation.

Internal variables

ArcCAD provides you with three internal variables that can be used in logical and arithmetic expressions.

$RECNO—the record number of a record in the selected data file.

$PI—the value for *pi* (3.14159...), which is the ratio of a circle's circumference to its diameter.

$E—the value for *e* (approximately 2.71828), which is the base of the number system for natural logarithms.

These internal variables can be used as operands anywhere within a logical or arithmetic expression; for example

```
"$RECNO GT 100"
```

This selection expression will find all records from the selection set whose record number is greater than 100.

Feature selection set manipulation functions

The following functions are used to create, modify, list and delete feature selection sets.

(fssadd *featname fss*)

This function adds a named feature to a feature selection set. The actual output of this function is dependent upon the value of *fss* provided. This function will return either a feature selection set or `nil`. When a new member is added to *fss*, it is added to all feature selection sets bound to *fss*. In other words, if *fss* is assigned to other variables, they also reflect the addition.

If *fss* does not evaluate to `nil`, this function adds *featname* to *fss*. If *featname* is successfully added to the feature selection set, the new feature selection set is returned. If *featname* cannot be added to the feature selection set, `nil` is returned. If *featname* is already a member of *fss*, the operation is ignored and the original feature selection set is returned. If *featname* and *fss* are not derived from the same theme, `nil` is returned.

If *fss* evaluates to `nil`, a new feature selection set is created. This new feature selection set will contain *featname* as its only member.

For example, to add the last feature in the theme 'ROADS' to an existing feature selection set named *fss1*, the following code fragment could be used:

```
; Get name of last feature in the theme.
(setq f1 (featlast “roads”))
;
; Add feature to feature selection set
(setq fs (fssadd f1 fss1))
(if (= fs fss1)
   (princ “feature added\n”)
   (princ “feature not added\n”)
)
```

(fssand *fss1 fss2*)

This function creates a new feature selection set that contains all of the feature names in both of the feature selection sets provided. If both *fss1* and *fss2* are derived from the same theme, a new feature selection set is created. If the feature selection sets are not derived from the same theme, `nil` is returned.

If the two feature selection sets do not have any members in common, the resulting feature selection set will not have any members. This function is equivalent to a Boolean AND of the two feature selection sets.

For example, if feature selection sets *fss1* and *fss2* are derived from the same theme, the following code fragment would return a new feature selection set that contains features common to both feature selection sets.

```
(setq fss3 (fssand fss1 fss2))
(if (null fss3)
   (princ "the operation failed\n")
   (princ "the operation was successful\n")
)
```

(fssclr *fss*)

This function removes all of the members from a feature selection set. If all members are successfully removed from *fss*, the name of the feature selection set is returned. If all of the members cannot be removed, `nil` is returned.

For example, to remove all of the members of the existing feature selection set named *fss1*, the following code fragment could be used.

```
(setq fs (fssclr fss1))
(if (= fs fss1)
   (princ "the operation was successful\n")
   (princ "the operation failed\n")
)
```

(fssdel *featname fss*)

This function removes a named feature from a feature selection set. If *featname* is successfully removed from *fss*, the name of the feature selection set is returned. If *featname* cannot be removed from the feature selection set, `nil` is returned. If *featname* is not a member of the feature selection set, `nil` is returned.

For example, to delete the last feature in the theme 'ROADS' from an existing feature selection set named *fss1*, the following code fragment could be used.

```
; Get name of last feature in theme
(setq f1 (featlast "roads"))
;
; Remove the feature from the feature
; selection set
```

```
(setq fs (fssdel f1 fss1))
(if (= fs fss1)
   (princ “feature successfully removed\n”)
   (princ “feature not removed\n”)
)
```

(fssfree *fss*)

This function frees resources allocated to a feature selection set. The resources allocated to *fss* is freed. This allows other feature selection sets to be created since there are a finite number of feature selection sets (128) available. This function always returns `nil`.

For example, to free the resources allocated to the feature selection set named *fss1*, the following code fragment could be used.

```
(fssfree fss1)
```

(fssget *theme state mode opt1 opt2*)

This function creates a feature selection set based upon the state of the selection and a selection method indicated by the user. This function will only process features that are members of *theme*. All other features will be ignored.

state: the argument *state* represents the current state of the selection set. By default, all the features of a given theme are selected. If the user wants to select a subset of features from a current selection set, the *state* of the selection would be “*C*” (current). If the user wants to select features from the entire database associated with the specified theme, the *state* of the selection would be “*A*” (all). In other words, if the *state* argument is “*C*”, this function behaves similarly to the **reselect** command. On the other hand, if the *state* argument is “*A*”, this function behaves similarly to the **aselect** command.

mode: there are currently nine different selection modes available to the user. A list of these modes is provided below.

“CC” *pt1 radius*	Selects all features crossing or inside a circle whose center is *pt1* and the radius is *radius*.
“CW” *pt1 radius*	Selects all features inside a circle whose center is *pt1* and the radius is *radius*.
“E” *expression nil*	Selects all features that satisfy a logical expression. *Expression* contains one or more valid logical

	expressions. For further details refer to the section on logical expressions in this chapter.
"P" *nil nil*	Returns the currently selected features of the theme. This option will always return the current selection for the theme and assumes that the *state* argument is always *"C"* (current). If the user specifies option *"A"* (all) as the *state* argument, the function ignores this option and assumes option *"C"* as the *state* argument. However, since the *state* argument is always evaluated by the function, the user should not skip this option or specify any other option keyword except *"C"* or *"A"*.
"PC" *list1 nil*	Selects all features crossing or inside a polygon whose vertices are stored in *list1*.
"PW" *list1 nil*	Selects all features inside a polygon whose vertices are stored in *list1*.
"WC" *pt1 pt2*	Selects all features crossing or inside a box whose opposite corners are *pt1* and *pt2*.
"WW" *pt1 pt2*	Selects all features inside a box whose opposite corners are *pt1* and *pt2*.
nil nil nil	Selects all features interactively. Only one mode of input is performed.

The following code fragment shows some examples of the **fssget** function. These examples assume that there is a valid polygon theme named LANDUSE with a user-defined attribute, named LUCODE, associated with each polygon in the database.

```
; Select features that cross or are within a user-
; defined window
(setq pt1 (getpoint “First corner: ”))
(setq pt2 (getcorner pt1 “Opposite corner: ”))
(setq fss1 (fssget “landuse” “a” “wc” pt1 pt2))
;
; Select features using a logical expression
(setq fss2 (fssget “landuse” “a” “e” “lucode = 2” nil))
;
; Select features interactively
(setq fss3 (fssget “landuse” “a” nil nil nil))
```

Warning: The results of each **fssget** operation are always stored as the theme's feature selection set. When this function is used, the base of selectable features is dependent on the value of the *state* argument. For instance, if you specify the *state* argument as "C", the base of selectable features is *the theme's feature selection set*. On the other hand, if you specify "A" as the *state* argument, the base of selectable features is *all features in the theme*. Therefore, the value of the *state* argument influences the results of the selection and should be used with care.

(fsslength *fss*)

This function returns the number of members contained in a feature selection set. If the specified feature selection set is valid and contains 0 or more members, the function returns a nonnegative real number. If the specified feature selection set is not a valid name, `nil` is returned.

For example, to print the number of features that are contained in a feature selection set created by the user, the following code segment could be used.

```
; Get theme from user
(setq theme (getstring "Theme name: "))
;
; Get user-defined feature selection set
(setq fss1 (fssget theme "a" nil nil nil))
;
; Get the number of members
(setq cnt 0)
(if (null fss1)
     (princ "error creating feature selection set\n")
     (setq cnt (fsslength fss1))
)
(princ "There are ")
(princ cnt)
(princ " members\n")
```

(fssmemb *featname fss*)

This function determines if a named feature is a member of a feature selection set. If *featname* is a member of *fss*, `T` is returned. If *featname* is not a member of the feature selection set, `nil` is returned.

For example, to determine if a feature named *feat1* is a member of the feature selection set named *fss1*, the following code fragment could be used.

```
(setq result (fssmemb feat1 fss1))
(if (null result)
   (princ "feature is not a member\n")
   (princ "feature is a member\n")
)
```

(fssname *fss index*)

This function returns the name of the indexed feature in the feature selection set. The *index* must always be specified as a nonnegative real number. The index of the first member of the feature selection set is 0.0. If *fss* contains a member at the *index* position, the feature name is returned. If *index* is negative or is greater than the number of members contained in the feature selection set, `nil` is returned.

For example, to retrieve the name of the sixth feature of a feature selection set named *fss1*, the following code fragment could be used.

```
; Remember that the first feature's index is 0.0
(setq feat1 (fssname fss1 5))
(if (null feat1)
     (princ "feat1 does not have 6 members\n")
     (princ "feature name retrieved\n")
)
```

(fssnot *fss*)

This function switches the members of a feature selection set. All of the current members of the feature selection set are replaced by all of the nonselected members of the associated theme. This function is equivalent to a Boolean complement of the feature selection set. If members of a feature selection set are successfully complemented, the feature selection set is returned. If the feature selection set cannot be complemented, `nil` is returned.

For example, to create a feature selection set that contains all of the features of the theme ROADS that are *not* selected by the user, the following code fragment could be used.

```
; Get user’s selection
(setq fss1 (fssget “roads” “a” nil nil nil))
(if (null fss1)
     (princ “feature selection operation failed\n”)
     (progn
     ;  Select everything not currently selected
        (setq fs (fssnot fss1))
        (if (= fs fss1)
           (princ “operation successful\n”)
           (princ “operation failed\n”)
        )
     )
)
```

(fssor *fss1 fss2*)

This function creates a new feature selection set that contains all of the members of both of the feature selection sets provided. If both *fss1* and *fss2* are derived from the same theme, a new feature selection set is created that contains members in both *fss1* and *fss2*. If the feature selection sets are not derived from the same theme, `nil` is returned. This function is equivalent to a Boolean OR of the two feature selection sets.

The following code fragment creates a feature selection set that contains a combination of features from two different feature selection sets. This example assumes that ‘LANDUSE’ is a valid polygon theme that has a user-defined attribute named ‘LUCODE’.

```
; Get first feature selection set from user
(setq fss1 (fssget “landuse” “a” nil nil nil))
;
; Get second feature selection set using a
; logical expression
(setq fss2 (fssget “landuse” “a” “e” “lucode = 2” nil))
;
(if (or (= fss1 nil) (= fss2 nil))
(princ “one of the feature selection operations
failed\n”)
      (progn
      ;  Combine the feature selection sets
         (setq fss3 (fssor fss1 fss2))
         (if (null fss3)
            (princ “operation failed\n”)
            (princ “operation successful\n”)
         );end if
      );end progn
);end if
```

(fssput *fss*)

This function replaces the theme's feature selection set with the feature selection set provided. If the replacement is successful, the name of the feature selection set is returned. If the replacement cannot be performed, `nil` is returned.

The following code fragment shows an example of the **fssput** function. In this example, several feature selection sets are created. Each time the **fssget** function is called, the theme's selection set is changed. This example shows how to change the theme's selection to a specific feature selection set. This example assumes that the theme ROADS exists.

```
; Get first feature selection set from the user
(setq fss1 (fssget “roads” “a” nil nil nil))
; the theme’s selection set is now identical to
; that of fss1
;
; Get second feature selection set from the user
(setq fss2 (fssget “roads” “a” nil nil nil))
; the theme’s selection set is now identical to
; that of fss2
;
; Change the theme’s selection set back to that of
; fss1
(setq fs (fssput fss1))
(if (= fs fss1)
     (princ “replacement successful\n”)
     (princ “replacement failed\n”)
)
```

(fssxor *fss1 fss2*)

This function creates a new feature selection set that contains all of the members that are not in both of the sets provided. If both *fss1* and *fss2* are derived from the same theme, a new feature selection set is returned. If the feature selection sets are not derived from the same theme, `nil` is returned. This function is equivalent to a Boolean exclusive OR of the two feature selection sets. If *fss1* is identically equal to *fss2*, an empty feature selection set is created.

For example, to create a feature selection set that contains all features that are not in both feature selection sets *fss1* and *fss2*, the following code fragment could be used.

```
(setq fss3 (fssxor fss1 fss2))
(if (null fss3)
   (princ “operation failed\n”)
   (princ “operation successful\n”)
)
```

Feature name functions

The following functions are used to retrieve feature names and entity names.

(entfeat *ename theme*)

This function returns the feature name(s) of features linked to the named entity. The *ename* must be the name of the main entity. This function searches *ename* and retrieves a list of the appropriate feature names that correspond to the entity. This function always returns the name of the main feature. You should use the **featnext** function to access the subfeatures of the main feature (if any exist). The actual output of this function is dependent upon the value of *theme*. This function will return either a list or `nil`.

If *theme* does not evaluate to `nil`, this function will check to see if *ename* is a member of *theme*. If the entity is a member of the theme, a list containing only the corresponding feature name is returned. If the entity is not a member of the theme, `nil` is returned.

If *theme* evaluates to `nil`, a list of all of the corresponding feature names is returned. If the entity is not a member of any theme, `nil` is returned.

For example, to determine the number of themes that a user-selected entity is linked to, the following code fragment could be used.

```
; Select the entity
(setq ent1 (car (entsel)))
; Get a list of themes
(setq list1 (entfeat ent1 nil))
(princ “Entity is a member of ”)
(princ (length list1))
(princ “themes\n”)
```

The following code fragment can be used to determine if a user-selected entity is linked to a feature in the theme ROADS.

```
; Select the entity
(setq ent1 (car (entsel)))
; Check for presence in theme
(setq list1 (entfeat ent1 “ROADS”))
(if (null list1)
   (princ “entity is not linked to theme”)
   (princ “entity is linked to theme”)
)
```

Notes

- In a polygon theme, links are established only to the label point(s) and, therefore, you must select the label point in order to retrieve the corresponding polygon feature. Selecting the polygon boundary always returns `nil`. You must use the **featnext** function to access the line (polygon boundary) subfeatures.

- *Tic*, *image* and *record* themes do not have entity-feature links in ArcCAD; therefore, the **entfeat** function on these themes always returns `nil`.

(featent *featname*)

This function returns the corresponding entity name of a feature. If *featname* is displayed in the drawing, the corresponding entity name is returned. If *featname* is not displayed in the drawing, `nil` is returned.

For example, *feat1* is the name of a feature that has been rendered in the drawing. To retrieve the entity's data from the drawing, the following code fragment could be used.

```
(setq ent1 (featent feat1))
(if (null ent1)
    (princ "feature not displayed in the drawing\n")
    (progn
        (princ "entity name has been retrieved\n")
        (entget ent1)
    )
)
```

(featlast *theme*)

This function returns the name of the last feature in a theme. This function can be used to retrieve the name of a feature that was created by an ArcCAD command. If *theme* contains features, the name of the last feature is returned. If *theme* does not contain any features, `nil` is returned.

The following example shows how to retrieve the name of a feature that was just added to the theme ROADS using the **addfeat** command.

```
Command: addfeat
Theme name (?/<theme>): roads
Select objects: 1 selected, 1 found
Select objects: <CR>
Optional property table (?/<none>): <CR>
1 Written to C:\ROADS, 0 duplicated, 0 ignored
```

The following code fragment can be used to retrieve the name of the feature created using the **addfeat** command above.

```
(setq featl (featlast “roads”))
```

(featnext *theme featname level*)

This function returns the name of the next feature in a theme. The actual feature name returned is a function of the values of *featname* and *level*.

If *featname* evaluates to `nil`, the value of *level* is ignored. The resulting name returned is that of the first feature defined in *theme*. If *theme* does not contain any features, `nil` is returned.

If *featname* does not evaluate to `nil`, the resulting feature name returned is dependent upon the current value of *featname* and *level*. *featname* should be set to the name of the feature whose next feature (subfeature) name is to be retrieved. If *theme* does not contain any features, `nil` is returned.

The following table shows the relationships between *featname*, *level*, and the resulting feature name returned by the function.

Current Feature Name	**Level 1**	**Level 2**	**Level 3**
Annotation	Next annotation	Next annotation	Next annotation
Line header	Next line header	Next vertex of line	Next vertex of line
Line vertex	Next line header	Next vertex of line or line end	Next vertex of line or line end
Line end	Next line header	Next line header	Next line header
Point	Next point	Next point	Next point
Polygon header	Next polygon	Next point or line header of polygon	Next point or line header of polygon
Polygon point	Next polygon	Next point or line header of polygon	Next point or line header of polygon
Polygon line header	Next polygon	Next line header of polygon	Next vertex of line
Polygon line vertex	Next polygon	Next line header of polygon	Next vertex of line or line end
Polygon line end	Next polygon	Next line header of polygon	Next line header of polygon
Polygon end	Next polygon	Next polygon	Next polygon
Record	Next record	Next record	Next record
Tic	Next tic	Next tic	Next tic

For example, to retrieve the name of the first vertex in the third-line subfeature of the first polygon feature in the theme LANDUSE, the following code fragment could be used.

```
;Get name of first polygon in theme
(setq feat1 (featnext “landuse” nil nil))
(if (null feat1)
    (progn
        (princ “no topology in theme\n”)
        (setq ok nil)
    )
    ( setq ok 1)
)
;Read until 3rd line is found
(setq cnt 0)
(while ok
    ;Get next subfeature which may be point or line
    (setq subfeat (featnext “landuse” feat1 2))
    (if (null subfeat)
        (progn
          (princ “polygon has no subfeatures\n”)
          (setq ok nil)
        )
        (progn
          ;Check feature to see if it is a line
          (setq list1 (featget subfeat nil))
          (if (= “LINE” (assoc 0 list1))
            (progn
              (setq cnt (1+ cnt))
              (if (= cnt 3)
                (setq ok nil)
              )
            )
          )
          (setq feat1 subfeat)
        )
    )
)
;Get first vertex of line feature
(if (= cnt 3)
    (progn
        (setq feat1 (featnext “landuse” subfeat 3))
        (if (null feat1)
          (princ “success\n”)
          (princ “failure\n”)
        )
    )
)
```

(featthm *featname*)

This function returns the name of the theme in which the named feature is a member. If *featname* is a valid feature name, the name of the theme that contains the named feature is returned. If *featname* is not a valid feature name, `nil` is returned.

For example, to determine the name of the theme that a user-selected entity is linked to, the following code fragment could be used.

```
; Select the entity
(setq ent1 (car (entsel)))
; Get the list of feature names
(setq list1 (entfeat ent1 nil))
(if (null list1)
     (princ “Entity is not a member of a theme\n”)
       (progn
       ; Get the name of the first theme
       (setq theme (featthm (car list1)))
       (princ “Entity is a member of the theme ”)
       (princ theme)
     )
)
```

Feature data functions

The following functions are used to create, modify, and delete features and feature data in a theme.

(featdel *featname*)

This function removes the named feature from its GIS data set. If *featname* is successfully removed from the GIS data set, the name of the feature is returned. If *featname* cannot be removed from the GIS data set, `nil` is returned.

This function invalidates all feature selection sets that are bound to the feature name. In addition, the entity feature link that is associated with the corresponding feature is dropped (if any exists).

For example, to remove a user-selected feature from a theme, the following code fragment could be used.

```
; Select the entity
(setq ent1 (car (entsel)))
; Get the list of feature names
(setq list1 (entfeat ent1 nil))
(if (null list1)
     (princ “Entity is not a member of a theme\n”)
     (progn
        ; Delete the feature from the theme
        (setq feat1 (car list1))
        (setq result (featdel feat1))
        (if (= feat1 result)
           (princ “feature successfully deleted\n”)
           (princ “feature deletion failed\n”)
        )
     )
)
```

(featget *featname itemlist*)

This function retrieves the named feature’s data from the theme. If *featname* exists, a list containing its data is returned. If *featname* does not exist, `nil` is returned. *itemlist* is a list containing item names. If *itemlist* does not evaluate to `nil`, the corresponding item values will be included in the list of feature data. See the *itemlist* option at the end of this function for further details.

The format of the information returned for each of the six different feature classes is shown below. The sublists may not be in the order shown. The **assoc** function can be used to retrieve a specific sublist regardless of its position in the list.

■ ANNOTATION features are returned in the following format. This format is styled after the TEXT listing from the **entget** function.

```
(     (-1 . <Entity name: 0>)
      (0 . “ANNOTATION”)    feature class
      (1 . “text”)          text string
      (6 . featname)        feature name (read only)
      (40 . hgt)       height
      (41 . wdtscl)         width scale
      (42 . level)          annotation level
      (43 . symbol)         symbol number
      (59 . Internal-ID)    Internal-ID (read only)
      (10  x1  y1)          1st position point
      (11  x2  y2)          2nd position point (optional)
```

```
     (12  x3  y3)           3rd position point (optional)
     (13  x4  y4)           4th position point (optional)
     (14  x5  y5)           1st arrow point (not used)
     (15  x6  y6)           2nd arrow point (not used)
     (16  x7  y7)           3rd arrow point (not used)
)
```

■ POINT features are returned in the following format. This format is styled after the POINT listing from the **entget** function.

```
(    (-1 . <Entity name: 0>)
     (0 . "POINT")          feature class
     (6 . featname)         feature name (read only)
     (10  x  y)        coordinates
     (40 . poly1)           polygon ID (read only)
     (58 . User-ID)         User-ID
     (59 . Internal-ID)     Internal-ID (read only)
)
```

■ TIC features are returned in the following format. This format is styled after the POINT listing from the **entget** function.

```
(    (-1 . <Entity name: 0>)
     (0 . "TIC")            feature class
     (6 . featname)         feature name (read only)
     (10  x  y)        coordinates
     (58 . User-ID)         User-ID
     (59 . Internal-ID)     Internal-ID (read only)
)
```

■ LINE features are returned in the following format. When the **featget** function is called with the name of the line feature, the following list is returned.

```
(    (-1 . <Entity name: 0>)
     (0 . "LINE")           feature class
     (6 . featname)         feature name (read only)
     (70 . vcnt)            number of vertices (read only)
     (40 . node1)           from-node (read only)
     (41 . node2)           to-node (read only)
     (42 . poly1)           left polygon (read only)
     (43 . poly2)           right polygon (read only)
     (58 . User-ID)         User-ID
     (59 . Internal-ID)     Internal-ID (read only)
)
```

When **featget** is called with the name of a line subfeature, the following list is returned.

```
(     (-1 . <Entity name: 0>)
      (0 . “VERTEX”)         subfeature class
      (6 . featname)         subfeature name (read only)
      (10  x  y)        coordinate
)
```

When **featget** is called with the name of the last line subfeature, the following list is returned to indicate the end of the feature.

```
(     (-1 . <Entity name: 0>)
      (0 . “LINEEND”) end of line marker
)
```

■ RECORD (dBASE) features are returned in the following format. When the **featget** function is called, the following list is returned.

```
(     (-1 . <Entity name: 0>)
      (0 . “RECORD”)         subfeature class
      (6 . featname)         subfeature name (read only)
      (59 . Internal-ID)     Internal-ID (read only)
)
```

To obtain item (feature attribute) values, the **featget** function should be used with the *itemlist* option.

■ POLYGON features are returned in the following format. When the **featget** function is called with the name of the polygon feature, the following list is returned.

```
(     (-1 . <Entity name: 0>)
      (0 . “POLYGON”) feature class
      (6 . featname)         feature name (read only)
      (58 . User-ID)         User-ID
      (59 . Internal-ID)     Internal-ID (read only)
      (70 . pcnt)            number of points (read only)
      (71 . lcnt)            number of lines (read only)
      (10  x  y)        coordinates of centroid (read only)
      (11  xmin  ymin)       box min coordinates (read only)
      (12  xmax  ymax)       box max coordinates (read only)
)
```

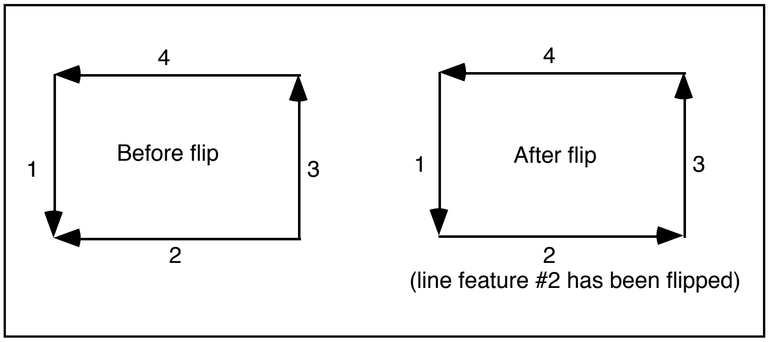
When processing a polygon feature, the points are always accessed before the lines and their vertices. When **featget** is called with the name of a polygon subfeature, one of the following lists is returned.

```
(     (-1 . <Entity name: 0>)
      (0 . "POINT")          feature class
      (6 . featname)         subfeature name (read only)
      (10  x  y)        coordinates
      (40 . poly1)           polygon ID (read only)
      (58 . User-ID)         User-ID
      (59 . Internal-ID)     Internal-ID (read only)
)

(     (-1 . <Entity name: 0>)
      (0 . "LINE")           feature class
      (6 . featname)         subfeature name (read only)
      (70 . vcnt)            number of vertices (read only)
      (71 . flip)            flip arc coordinates (optional)
      (72 . island)          island in a polygon (optional)
      (40 . node1)           from-node (read only)
      (41 . node2)           to-node (read only)
      (42 . poly1)           left polygon (read only)
      (43 . poly2)           right polygon (read only)
      (58 . User-ID)         User-ID
      (59 . Internal-ID)     Internal-ID (read only)
)

(     (-1 . <Entity name: 0>)
      (0 . "VERTEX")         subfeature class
      (10  x  y)        coordinate
      (6  . featname) Sub-subfeature name
)

(     (-1 . <Entity name: 0>)
      (0 . "LINEEND") end of line marker
)
(     (-1 . <Entity name: 0>)
      (0 . "POLYEND") end of polygon marker
)
```

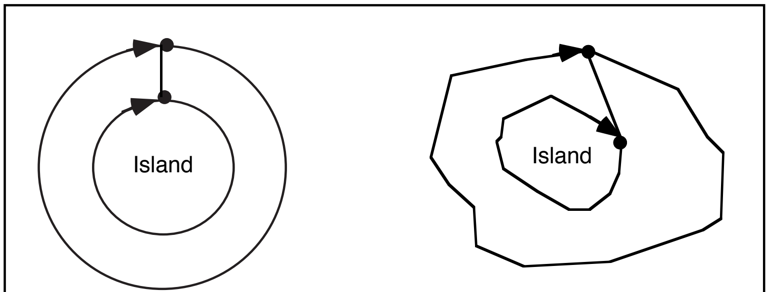
Remember that polygon features have subfeatures called points and lines and sub-subfeatures called vertices. Also remember that the first polygon feature in a polygon theme is always the *universe polygon*. For additional details, refer to the *ArcCAD User's Guide*.

Note: The two group codes **`71`** (flip) and **`72`** (island) mentioned in the line header of a polygon deserve further explanation:

> **`(71 . flip)`** this code is optional and is used to represent flip line coordinates. The value **1** indicates that the line's coordinates should be flipped, and the default value **0** represents *no flip*. The following figure demonstrates the usage of the **flip** option.



> **`(72 . island)`** this code is optional and is used to indicate that the *remaining* line (sub) features of the polygon are part of an island within the current polygon. The following figure demonstrates the potential use of the **island** option in a polygon theme:



Island formation in polygon themes

itemlist option

If the *itemlist* option does not evaluate to `nil`, the **featget** function retrieves both feature data and feature attribute values from the specified theme. The list of attribute data is appended to the end of the list normally returned with the main features. This new list will have the following format:

```
(-3                                  item flag
     (“ESRI”                         application name
          (1040 . real)              real value1 (optional)
          (1000 . “text”) character value1 (optional)
          (1040 . real)              real value2 (optional)
          (1000 . “text”) character value2 (optional)
          .....
          .....
          (1040 . real)              real valueN (optional)
          (1000 . “text”) character valueN (optional)
     )
)
```

Only one of the two possible group codes is present. The storage type of the item determines which group code is output.

This list is only returned when *featname* is the name of a main or complex feature. To illustrate, consider the following example where **featget** is used with the *itemlist* option to retrieve the coordinates of a line feature and its corresponding attribute values. When the function is called with the name of the line feature with the *itemlist* option, the following list is returned.

```
(    (-1 . <Entity name: 0>)
     (0 . “LINE”)                    feature class
     (6 . featname)                  feature name (read only)
     (70 . vcnt)                     number of vertices
     (40 . node1)                    from-node
     (41 . node2)                    to-node
     (42 . poly1)                    left polygon
     (43 . poly2)                    right polygon
     (58 . User-ID)                  User-ID
     (59 . Internal-ID)              Internal-ID (read only)
     (-3                             item flag
       (“ESRI”                       application name
          (1040 . real)              real value1 (optional)
          (1000 . “text”) character value1 (optional)
          (1040 . real)              real value2 (optional)
          (1000 . “text”) character value2 (optional)
          .....
          .....
          (1040 . real)              real valueN (optional)
          (1000 . “text”) character valueN (optional)
       )
     )
)
```

When the **featnext** function is called with the name of the line feature, the name of the first subfeature (vertex) is returned. When **featget** is called with the name of the first subfeature, the following list is returned.

```
(     (-1 . <Entity name: 0>)
      (0 . "VERTEX")              subfeature class
      (10  x  y)             coordinate
      (6 . featname)              subfeature name
)
```

You will notice that the list returned from the function does not contain any reference to item data because the feature retrieved is not a main feature.

(featmake *theme featlist itemlist*)

This function creates a new feature in a theme. If the feature is successfully created, the name of the feature is returned. If the feature cannot be created, `nil` is returned.

If *itemlist* evaluates to `nil`, the corresponding item values for the feature are ignored. If *itemlist* does not evaluate to `nil`, the item values for the feature are retrieved. Values will not be returned for item names that are not in *itemlist*. In addition, the values that are returned are in the same order as their item names.

Featlist is assumed to be in the same format as that returned by the **featget** function. The only exception to this is when polygon features are being created.

A polygon theme is composed of lines and points. The actual polygon features are not created until the theme's polygon topology has been created. To create a polygon theme, add points and line features to the theme. Calls to the **featlist** function will return `nil` until the theme is processed as mentioned earlier. In addition, **featmake** will return a dummy feature name.

When creating line features, multiple calls to **featmake** are required. Each of the subsequent calls defines the vertex subfeatures. When all of the vertex subfeatures have been defined, a call to **featmake** with the following list must be used to terminate the feature.

```
(     (-1 . 0)
      (0 . "LINEEND")
)
```

To illustrate the creation of a complex feature, consider the following code fragment, which creates a line feature with four vertices. The coordinates of the vertices are 0,0 1,1 2,0 and 3,1. Additionally, the User-ID of the line is 123. This example assumes that *linetheme* is a valid line theme.

```
(setq list1
     (list   (cons -1 0)
             (cons 0 “LINE”)
             (cons 40 0.0)
             (cons 41 0.0)
             (cons 42 0.0)
             (cons 43 0.0)
             (cons 58 123.0)
     )
)
(featmake linetheme list1)  ;create main feature
(setq list1
     (list(cons -1 0)
             (cons 0 “VERTEX”)
             (list 10  0.0  0.0)
     )
)
(featmake linetheme list1)  ;create 1st subfeature
(setq list1
     (list(cons -1 0)
             (cons 0 “VERTEX”)
             (list 10  1.0  1.0)
     )
)
(featmake linetheme list1)  ;create 2nd subfeature
(setq list1
     (list(cons -1 0)
             (cons 0 “VERTEX”)
             (list 10  2.0  0.0)
     )
)
(featmake linetheme list1)  ;create 3rd subfeature
(setq list1
     (list(cons -1 0)
             (cons 0 “VERTEX”)
             (list 10  3.0  1.0)
     )
)
(featmake linetheme list1)  ;create 4th subfeature
(setq list1
     (list(cons -1 0)
             (cons 0 “LINEEND”)
     )
)
(featmake linetheme list1)    ;close feature
```

To illustrate the creation of a normal feature, consider the following example, which creates a point feature at 10,10 with a User-ID of 101. Please note that this example also modifies the User-ID in the attribute file. This example assumes that *pointtheme* is a valid point theme with a valid attribute file.

```
(setq list1
     (list   (cons -1 0)
             (cons 0 "POINT")
             (list 10  10.0  10.0)
             (cons 40 0.0)
             (cons 58 101.0)
             (list   -3
                  (list   "esri"
                      (cons 1040 101.0)
                  )
             )
     )
)
;Setup list of item names
(setq itemlist (list "User_ID"))
;Create feature
(featmake pointtheme list1  itemlist)
```

In the above example, it is not necessary to call the **featmake** function with a list containing 'LINEEND' because point features do not contain any subfeatures.

(featmod *featlist itemlist*)

This function modifies the data of a feature or subfeature. If the feature is successfully modified, the name of the feature is returned. If the feature cannot be modified, `nil` is returned. The name of the feature to be modified must be found in the group code 6 of *featlist*.

If *itemlist* evaluates to `nil`, all item values in *featlist* are ignored. If *itemlist* does not evaluate to `nil`, the item values for the feature are modified. The values of items are not modified for item names that are not in the *itemlist*. In addition, the values are assumed to be in the same order as their item names. The format of the *itemlist* is assumed to be in the same format as returned by the **featget** function with the *itemlist* option.

Featlist is assumed to be in the same format as that returned by the **featget** function. The only exception to this occurs when a polygon feature is to be modified.

The main features of polygons cannot be modified. Only the subfeatures of a polygon can be modified. The lists in the documentation for **featget** show which group codes are modifiable and which group codes are read-only for line and point subfeatures of polygons. Also remember that the first polygon in a polygon theme is the *universe polygon* and you should never modify it.

It is important to note that **featmod** cannot be used to change the number of vertices in a line feature. **Featmod** also cannot be used to modify the name of items or the item's type in a record theme.

For example, to change the User-ID of the line feature *feat1*, the following code fragment can be used.

```
; Get new User-ID
(setq newid (getreal "New user id: "))
;
; Get topology only
(setq list1 (featget feat1 nil))
;
; Get current value
(setq old (assoc 58 list1))
(setq new (cons 58 newid))
;
; Substitute value in array
(setq newlist (subst new old list1))
;
   ; Modify line
(setq result (featmod newlist))
(if (null result)
   (princ "modification failed\n")
   (princ "modification successful\n")
)
```

Chapter 3

Item access

Item functions are used to manipulate dBASE RECORD themes. They are also used to manipulate the attribute files of POLYGON, LINE, POINT, and TIC themes. They perform the basic operations of creating, listing, and retrieving item definitions. These functions cannot be used to modify the item values. The functions described in the 'Feature selection set manipulation functions' section (**featget**, **featmake**, and **featmod** functions) can be used to modify the values associated with the items.

As explained in the introduction chapter, some feature classes have implied attribute files. The following list shows these implications.

Feature class	**Attribute file**
Annotation	None
Line	AAT
Point	PAT
Polygon	PAT
Record	User defined
TIC	TIC

In other words, when you specify the *theme* name in the following item manipulation functions, the software automatically performs the operation on the implied attribute files. For example, if you use the **itmdef** function (see next section for details) with a polygon theme, the function automatically defines the item in the theme's implied polygon attribute table (PAT) file.

Note: You cannot use item manipulation functions on an annotation theme. This is because annotation features are not spatially related to other features and therefore do not carry any implied attribute table. Annotation themes in ArcCAD® software are used only for annotating the geographic features.

The item access functions described in the following section let you define dBASE RECORD themes (dBASE files) and add item definitions to store attribute information. For example, the **itmdef** function lets you define items. Similarly, the **itmexi** function checks for the existence of the item in the theme's data file prior to writing or retrieving item values. The **itmget** function lets you access the item definition, and the **itmlst** function lists all the item names in a theme's data file.

The following are some of the limitations that apply to item access functions:

- The maximum length of an item name is 10 characters.
- The only legal characters in item names are alphabetic characters, numbers and the underscore character.
- Item names cannot begin with a number.

Item manipulation functions

The following functions can be used to define, modify and list item definitions stored in database files.

(itmdef *theme list*)

This function adds a new item to a theme. If the item is successfully created, the name of the item is returned. If the item cannot be created, `nil` is returned. The function does not perform any user interaction. All of the parameters must be supplied. If user interaction is desired, use the **cmd** function to invoke the user command that defines an item.

There are currently four item types available: *Character*, *Date*, *Integer* and *Numeric*.

The format for the input list is

```
(  (-1 . 0)
   (0 . "ITEM")
   (1 . name)          item name
   (2 . type)          item type
   (70 . column)       start column
   (71 . width)        item width
   (72 . ndec)         number of decimal places
   (73 . owidth)       output width
)
```

For example, to define a record theme named 'LANDUSE' with items named 'AREA' and 'LANDUSE_ID', similar to that of a polygon theme, the following code fragment could be used.

```
; Define the theme
(setq list1
   (thmdef
      (list
         (cons -1 0) (cons 1 "landuse")
         (cons 2 "record") (cons 3 "c:\\path\\landuse")
         (cons 60 0)
      )
   )
)
(if (null list1)
    (princ "Theme definition failed\n")
    (progn
       (setq itm nil)
; Define item area
      (setq itm1
        (itmdef “landuse”
          (list
            (cons -1 0) (cons 0 “ITEM”)
            (cons 1 “area”) (cons 2 “N”)
            (cons 70 1) (cons 71 13)
            (cons 72 6) (cons 73 13)
          )
        )
      )
      (if (null itm1)
        (setq itm 1)
      )
; Define item landuse_id
      (setq itm1
        (itmdef “landuse”
          (list
            (cons -1 0) (cons 0 “ITEM”)
            (cons 1 “landuse_id”) (cons 2 “I”)
            (cons 70 38) (cons 71 11)
            (cons 72 0) (cons 73 11)
          )
        )
      )
      (if (null itm1)
        (setq itm 1)
      )
      (if (= itm 1)
        (princ “Item definition failed\n”)
        (princ “Item definition successful\n”)
      )
    )
)
```

(itmexi *theme itmname*)

This function checks for the existence of an item in a theme. If *itmname* exists in *theme*, `T` is returned. If *itmname* does not exist in *theme*, `nil` is returned.

For example, to check for the existence of an item in a theme, the following code fragment could be used:

```
; Get theme name
(setq theme (getstring “Name of theme: ”))
; Get item name
(setq item (getstring “Name of item: ”))
; Check for presence in theme’s attribute file
(setq result (itmexi theme item))
(if (null result)
   (princ “Item not defined in theme\n”)
   (princ “Item is defined in theme\n”)
)
```

(itmget *theme itmname*)

This function returns the definition of an item. If *itmname* exists in *theme*, a list containing the item’s type, width, and number of decimal places is returned. If *itmname* does not exist in *theme*, `nil` is returned.

The list returned is in a format that can easily be manipulated using the **assoc** function. The format of the returned list is

```
(  (-1 . <Entity name: 0>)
   (0 . “ITEM”)
   (1 . name)
   (2 . type)
   (70 . column)
   (71 . width)
   (72 . ndec)
   (73 . owidth)
)
```

For example, to retrieve the definition of the item named ‘AREA’ from a polygon theme named ‘SAMPLE’, the following code fragment can be used.

```
(itmget “sample” “area”)
```

This example would return the following list:

```
(  (-1 . <Entity name: 0>)
   (0 . “ITEM”)
   (1 . “AREA”)
   (2 . “N”)
   (70 . 1)
   (71 . 13)
   (72 . 6)
   (73 . 13)
)
```

(itmlst *theme [type]*)

This function returns a list of the items defined in a theme’s attribute table. If the table does not contain any items, `nil` is returned. The *type* argument is optional and is used to return only items of specific types.

Four item types are available: *Character*, *Date*, *Integer* and *Numeric*. Other item types available in dBASE® but not fully supported by ArcCAD include *Logical* (logical data type) and *Memo* (MEMO field).

Note: ArcCAD functions do not currently support MEMO fields.

The *type* argument is an integer (bit coded) with values as shown below:

Bit value	Meaning
1	List *numeric* items
2	List *integer* items
4	List *character* items
8	List *date* items
16	List *logical* items
32	List *memo* fields

The bit values can be added in any combination to return a variety of item types. For example, if you wish to list *numeric* and *character* items, you can set the value of *type* to 5.

Caution: Future releases of ArcCAD might use additional **itmlst** control bits, so avoid setting bits in your applications that aren’t shown in the above table.

Notes

■ If the specified theme is currently related to another theme (established using the **relate** or **ddrelate** command), the related theme's items are also displayed. Items in the related theme are prefixed with a pound sign (#).

■ The **itmlst** function cannot be used on *annotation* or *image* themes because these feature classes do not have corresponding database files.

The following code fragments show some uses of the **itmlst** function with the *type* argument. These examples assume that a theme named PARCELS exists in the current drawing and contains two user-defined items, OWNER (a *character* item storing the owner's name) and DATE (a *date* item storing the parcel registration date), in addition to the standard polygon attribute table (PAT) items AREA, PERIMETER, PARCELS_ and PARCELS_ID.

Example 1

List all the items in the theme's feature attribute table.

The following code fragment

```
(itmlst “parcels”)
```

would return the following list:

```
(“AREA” “PERIMETER” “PARCELS_” “PARCELS_ID” “OWNER” “DATE”)
```

Example 2

List only the *character* and *date* items:

The following code fragment

```
(itmlst “parcels” (+ 4 8))
```

would return the following list:

```
(“OWNER” “DATE”)
```

Example 3

Assume that the PARCELS theme is temporarily related to a *record* theme containing three *character* fields named STREET (street address), CITY (city name) and ZIP (five digit ZIP code).

The following code fragment

```
(itmlst “parcels” 4)
```

would return the following list:

```
(“OWNER” “#STREET” “#CITY” “#ZIP”)
```

Note that the items prefixed with a pound sign (#) represent items in the related theme.

Chapter 4

SQL access

The ArcCAD® SQL functions allow the AutoLISP® programmer to access tabular data stored in external relational database management systems (RDBMSs). The underlying method used for SQL access is the AutoCAD® ASI SQL engine. To allow RDBMS tables to be used by ArcCAD SQL functions, you must therefore configure the ASI environment (similar to configuring AutoCAD for ASE).

The SQL functions (as well as ASI and ASE) use SQL2 to access tables. This system defines a hierarchy that includes environments, catalogs, schemas and tables. The environment is the top level of the hierarchy and defines the use of a database program. A catalog is comparable to a database or a collection of database tables. The schema is a subset of a catalog that contains a portion of the complete database. A table contains the rows and columns of the data.

Refer to the *ArcCAD Installation Guide* and the *AutoCAD Installation Guide* for complete information about configuring the ASI environment.

The functions described in this section allow the management and editing of records in SQL tables. SQL tables are accessed using *connections* and *cursors*. A connection defines the SQL environment, as specified in ASI.INI, to be accessed. A cursor is a handle to a programmer-defined selection set in an SQL table. Before any SQL table can be accessed, you must open a connection to the appropriate SQL environment using **esri_cnnopn**. The specified environment must be configured in the AutoCAD ASI.INI file. After a connection is established, a cursor must be created to access an SQL table in that environment. Cursors are created using the **esri_curopn** function. The query string used when defining a cursor is an SQL select statement that defines the catalog, schema, table and the set of rows and columns in that table to access. Once a cursor has been created, you can query the cursor for information about the table using **esri_curcollst**, **esri_curcolexi**, **esri_curcoltype**, and **esri_curcolwid**, and column values may be retrieved and updated using functions such as **esri_curfirst**, **esri_curnext**, **esri_curabs**, **esri_curcolval** and **esri_curcolset**. When finished using the cursor and connection, you must release their resources by calling **esri_curcls** to close the cursor and **esri_cnncls** to close the connection.

The following simple example demonstrates how the ArcCAD SQL functions can be used to display a Microsoft Access® table:

```
; Assuming the database environment odbc_access is
; configured in ASI.INI, open a connection to it
(setq conname (esri_cnnopn “odbc_access” “john” “”))

; Open a cursor pointing to all records from the
; table called products
(setq mycursor “cur”)
(esri_curopn cur conname “select * from .”c:\msoffice\
 access\sampapps\nwind”.”products” nil nil)

; print the column names for the cursor
(setq collist (esri_curcollst mycursor))
(princ collist)

; print the column definitions
(foreach itm collist
   (setq typ (esri_curcoltyp mycursor itm))
   (setq wid (esri_curcolwid mycursor itm))
   (princ “Name: “) (princ itm)
   (princ “Type: “) (princ typ)
   (princ “Width: “) (princ wid)
   (princ “\n”)
)

; display all records of the table
(setq r (esri_curnext mycursor))
(while r
   (foreach itm collist
      (setq v (esri_curcolval mycursor itm))
      (princ v)
      (princ “ “)
   )
   (princ “\n”)
   (setq r (esri_curnext mycursor))
)

; close everything down
(esri_curcls mycursor)
(esri_cnncls conname)
```

SQL access functions

(esri_cnnactive *connectionname*)

This function checks if *connectionname* is an active SQL connection. If *connectionname* is active, `T` is returned. If *connectionname* is not active or is invalid, `nil` is returned.

(esri_cnncls *connectionname*)

This function closes the active connection *connectionname*. If *connectionname* is active, it is closed and `T` is returned. If *connectionname* is not active or is invalid, `nil` is returned.

(esri_cnnexi *connectionname*)

This function checks if the connection name *connectionname* exists in the ASI.INI file. If *connectionname* exists, `T` is returned. If *connectionname* does not exist, `nil` is returned.

(esri_cnnlst *status*)

This function returns a list of available connection names. The *status* option is used to specify the type of list created. Possible values for *status* are

1—return a list of active connection names.
2—return a list of inactive connection names.
3—return a list of both active and inactive connection names.

(esri_cnnopn *connectionname username password*)

Opens and activates a connection to an SQL2 environment. *connectionname*: specifies the connection as an environment within ASI.INI. *username* is an assigned name for logging into the database management system (DBMS) specified by *connectionname*. If a login name is not required for connection to the DBMS, set *username* to a null string. *password* is the password for user name. If the connection to the DBMS does not require a user name, or your login does not require a password, *password* should be set to a null string.

(esri_curabs *cursorname recnum*)

This function sets the current record of the active cursor *cursorname* to the record *recnum*. **esri_curabs** only works with scrollable cursors (i.e., cursors that were created with their scrolling option set to true [see **esri_curopn**]). *recnum* can be any record within the SQL data set. If the cursor is positioned correctly, this function returns `T`, otherwise, it returns `nil`. This function is used to randomly access records in an SQL data set.

(esri_curcls *cursorname [commit]*)

This function closes the cursor *cursorname*. If the cursor is successfully closed, `T` is returned. If the cursor cannot be closed or *cursorname* is invalid, `nil` is returned. The optional *commit* argument is used to commit any changes to the SQL database attached to *cursorname* made using **esri_curcolset**. Setting *commit* to `T` commits any changes. Setting *commit* to `nil` abandons any changes. *commit* only applies changes to the database if the cursor *cursorname* is sequential (nonscrollable).

(esri_curcollst *cursorname*)

This function returns a list of column names for the cursor *cursorname*. If *cursorname* is not a valid cursor or there are no columns, `nil` is returned.

(esri_curcolexi *cursorname columnname*)

This function checks if the column *columnname* exists in the database table pointed to by *cursorname*. If *columnname* exists in *cursorname*, `T` is returned, otherwise `nil` is returned.

(esri_curcolset *cursorname columnname newvalue*)

This function sets the value of the column *columnname* in cursor *cursorname* to *newvalue* for the current record. *newvalue* may be an AutoLISP string, integer, real number or nil.

(esri_curcoltype *cursorname columnname*)

This function returns a code that identifies the column type of column *columnname* in cursor *cursorname*. One of following codes is returned:

C	-	character
D	-	date
M	-	memo
I	-	short integer number
L	-	logical
B	-	SQL long integer number
Z	-	SQL double-precision number
S	-	SQL short integer number
R	-	SQL real number
E	-	SQL decimal number
P	-	SQL floating point number
W	-	SQL numeric
nil	-	*columnname* does not exist

(esri_curcolwid *cursorname columnname*)

This function returns the width of the specified column *columnname* in cursor *cursorname*.

(esri_curcolval *cursorname columnname*)

This function returns the value of the specified column *columnname* for the current record. *cursorname* is the name of the SQL cursor to use. If *cursorname* and *columnname* are valid, the value of *columnname* for the current record is returned. If *cursorname* or *columnname* is invalid, `nil` is returned. This function is used to retrieve column values from the table pointed to by a cursor.

Note that `nil` is also returned if *cursorname* and *columnname* are valid and the value of the column is NULL.

(esri_curexi *cursorname*)

This function checks for the existence of a cursor called *cursorname*. If *cursorname* exists, `T` is returned. If *cursorname* does not exist, `nil` is returned.

(esri_curfirst *cursorname*)

This function sets the current record of the cursor *cursorname* to the first record in the dataset. `T` is returned if the cursor was positioned correctly. `nil` is returned if the cursor could not be positioned correctly. The cursor *cursorname* must be a random-access (scrollable) cursor.

(esri_curlst)

This function returns a list of all currently valid cursor names.

(esri_curnext *cursorname*)

This function sets the current record of the cursor *cursorname* to the next record in the data set. `T` is returned if the cursor was positioned correctly. `nil` is returned if the cursor could not be positioned correctly. This function can be used with both sequential and random-access cursors.

(esri_curopn *cursorname connectionname query access update*)

This function creates the cursor *cursorname* attached to the SQL connection *connectionname*. The cursor contains the table records defined in the SQL query string *query*. If the cursor is successfully created, `T` is returned. If the cursor cannot be created, `nil` is returned.

query is an SQL select expression that defines the set of SQL records to attach to this cursor. It usually has the following form:

```
select fieldnames from catalog.schema.table
```

but the exact syntax is dependent on the DBMS to which you are connected. Refer to your AutoCAD documentation for complete information about supported SQL syntax.

access determines the access method for this cursor. SQL record access can be either sequential (also called nonscrollable) or random (scrollable). Sequential access limits you to stepping through the records in order, one at a time, from beginning to end. The advantage of a sequential cursor is that you can update the column values using **esri_curcolset**. Random-access cursors allow you to move through the SQL data set selecting records in any order. You cannot update column values using random-access cursors. Set *access* to `T` to create a random-access cursor or to `nil` to create a sequential-access cursor.

update sets the update privilege of the cursor when the original data set has been modified. If *update* is set to `T`, any changes made to the cursor are not immediately updated. The cursor must be closed and reopened to display changes in the SQL data set. If update is `nil`, changes made to the cursor are immediately reflected in the cursor.

Chapter 5

Additional functions

(esri_exec *command*)

This function is only available under the Windows version of ArcCAD and is used to execute other Windows applications from ArcCAD. **esri_exec** takes a single character string argument *command* which is the name of the program to execute, plus its associated arguments. You must specify the file extension for the program name (e.g., notepad.*exe*). Valid file extensions are .EXE, .COM and .PIF. The function will search for the executable program in the following directories in this order: current directory, the Windows directory, the Windows system directory, and finally, the directories specified in the DOS PATH environment variable.

For example

```
(esri_exec “notepad.exe c:\\config.sys”)
```

would start the Windows notepad program and load the CONFIG.SYS file into it.

If **esri_exec** successfully launches the application, the function returns `T`. If an error is encountered when executing the command, **esri_exec** returns one of the following error codes:

Error Number	Meaning
0	Not enough memory to execute program
1	File not found
3	Could not execute program

(arcadver)

This function returns a string that contains the current ArcCAD® release number. Applications can tell what release of ArcCAD is running by examining the string returned by **arcadver**.

For example

```
(arcadver)
```

might return the string:

```
“11.40”
```

(dirlst *[path type wildcard]*)

This function returns a list of a combination of files, subdirectories and coverages in the current (or in a specified) directory. The *path*, *type* and *wildcard* arguments are optional. If the optional arguments are omitted, **dirlst** returns a list of all files and subdirectories, including coverages, in the current working directory. Note that you can either ignore all the optional arguments or supply all of them. You cannot omit only one of the arguments. Consider the following example:

The following code fragment

```
(dirlst)
```

is identical to

```
(dirlst “.” (+ 1 2 4) “*”)
```

However, the following code fragment

```
(dirlst “.” (+ 1 2 4))
```

is invalid. You must supply `‘*’` as a wildcard string to complete the argument list.

path: This optional argument is a valid pathname to the directory to search. A drive letter is permitted in the path, and you can use the forward slash instead of the backslash (but remember that you must use \\ to obtain one backslash in a string). A period (`‘.’`) represents the current working directory.

type: This optional argument is a bit-coded integer value that filters the type of files to return. The following table explains the bit values and the data returned with each:

Bit value	Meaning
1	Return file names
2	Return subdirectory names
4	Return coverage names

The bit values can be added in any combination to return a variety of file types. For example, to return only coverage names and the file names in the specified directory, set the value of *type* to 5 (and set the *wildcard* argument to `'*'`).

wildcard: The optional *wildcard* argument can be used to further filter the returned list. Only objects matching the wildcard pattern are returned. In the pattern, alphabetic characters and numerals are treated literally, a question mark (?) matches a single character, an asterisk (*) matches a sequence of characters, and certain other characters have special meanings within the pattern. Any valid AutoCAD wildcard string is accepted. Refer to the **wcmatch** function in the *AutoCAD Customization Guide* for more details on *wildcard* options.

(arc_filecpy *source destination*)

This function copies one or more files to a specified directory. The *path* argument can specify more than one file by using wildcards.

The following code fragment copies the config.sys file to the c:\temp directory:

```
(arc_filecpy "c:\\config.sys" "c:\\temp")
```

The next code fragment copies all files with the .EXE file extension to a specified directory:

```
(arc_filecpy "c:\\programs\\*.exe" "c:\\newprogs")
```

source: This argument is a pathname specifying the file(s) to be copied. A drive letter is permitted in the path, and you can use the forward slash instead of the backslash (but remember that you must use \\ to obtain one backslash in a string). A period (`'.'`) represents the current working directory. A wildcard pattern may be used. In the pattern, alphabetic characters and numerals are treated literally, and an asterisk (*) matches a sequence of characters. Use "*.*" to copy all files in the source path directory.

destination: This argument is a pathname specifying where the files will be copied. The pathname must already exist.

(arc_filedel *filespec*)

This function deletes one or more files in a specified directory. The *filespec* argument can specify more than one file by using wildcard specifiers.

The following code fragment deletes the config.sys file:

```
(arc_filedel "c:\\config.sys")
```

The next code fragment deletes all files with the .EXE file extension in a specified directory:

```
(arc_filedel "c:\\programs\\*.exe")
```

filespec: This argument is a pathname specifying the file(s) to be deleted. A drive letter is permitted in the path, and you can use the forward slash instead of the backslash (but remember that you must use \\ to obtain one backslash in a string). A period (`'.'`) represents the current working directory. A wildcard pattern may be used. In the pattern, alphabetic characters and numerals are treated literally and an asterisk (*) matches a sequence of characters. Use "*.*" to delete all files in the specified directory.

Theme recovery functions

Whenever a drawing is loaded, ArcCAD checks to see if the drawing contains themes. If the drawing does contain themes, ArcCAD must verify that the geographic information system (GIS) data set and link file directory for each theme exist and are valid. If the GIS data set of any theme cannot be found, ArcCAD must either locate the data set or drop the theme. This process is called theme recovery. When a theme's data set needs to be recovered, by default, ArcCAD will present a dialog box into which the user can type the pathname to the missing GIS data set (or the SQL login and query for SQL record themes). If you want to automate this process for your users, control which themes are recovered, or perform special processing, you can use the following set of AutoLISP® functions.

(esri_userrepair *theme type suggestions*)

When a theme needs to be recovered and the theme's GIS data set needs to be located, ArcCAD first checks to see if there is an AutoLISP function defined with the name **esri_userrepair**. If this function is defined, ArcCAD calls this function rather than the default theme recovery function (**esrid_thmrecov** or **esrid_thmrecovsql**—see below), allowing you to intercept the theme recovery process. The argument *theme* identifies the name of the theme to recover, *type* is the theme type and *suggestions* is a list of character strings containing possible pathnames to the GIS data set for *theme*. Inside **esri_userrepair**, you can call the following function to reestablish the theme's data set:

(esri_thmrepair *theme dataset*)

where *theme* is the name of the theme to be repaired and *dataset* is the full pathname to its new GIS data set. When repairing SQL record themes, the syntax of this function is:

(esri_thmrepair *theme conn user pass query*)

where *theme* is the name of the theme to be repaired, *conn* is the SQL connection name, *user* is the user name, *pass* is the password and *query* is the SQL query expression.

If, instead, you want to drop one or more of the themes you are attempting to recover, you can call

(esri_thmdrop *themes*)

where *themes* is a list of theme names you want to drop.

If you want to let ArcCAD handle theme recovery for a particular theme or themes, call the default theme recovery handler **esrid_thmrecov** or **esrid_thmrecovsql**.

(esrid_thmrecov *theme type suggestions*)

This function displays a dialog box allowing the user to specify the location of a theme's missing GIS data set. The argument *theme* is the name of the theme to recover, *type* is the theme type and *suggestions* is a list of strings containing possible pathnames to the GIS data set for *theme*. This function cannot be used to recover SQL record themes. To reestablish the database for SQL record themes, use **esrid_thmrecovsql**.

(esrid_thmrecovsql *theme type suggestions*)

This function displays a dialog box allowing the user to specify the data set parameters for an SQL record theme. The argument *theme* is the name of the SQL record theme to recover, *type* is the theme type (SQL) and *suggestions* is not used and should be set to `nil`.

GUI access

ArcCAD GUI functions

ArcCAD provides a dialog box interface as an alternative to the command line interface. The use of some of these ArcCAD® dialog boxes has been extended to provide a set of GUI functions that may be invoked through AutoLISP. Note that you cannot invoke these dialog boxes through the ArcCAD **cmd** function.

There are two classes of dialog boxes in ArcCAD: browsers and function panels. Browsers are the low-level dialog boxes defined in ArcCAD that aid in the selection or display of objects from a list. These browsers are used as standard widgets in function panels. Function panels are comprehensive dialog boxes that are used to execute several related commands. These panels group functions are frequently used together in one dialog box.

The graphic user interface (GUI) functions described in this section are the low-level tools used to invoke the browsers to display or select an object from a list. For example, the **arcd_theme** function invokes a dialog box with a list of valid theme names in the current drawing. The GUI functions always start with an 'arcd_' prefix to differentiate them from other AutoCAD® and ArcCAD AutoLISP® functions.

Source code

The source code for the browsers and function panels is supplied with the software. Two DCL files included with ArcCAD are *pickers.dcl* and *ddarc.dcl*. These files can be found in the \ARCAD\BIN directory.

The file *pickers.dcl* contains the DCL definitions for the low-level browsers described in this section. You should never modify *pickers.dcl*. An error in *pickers.dcl* breaks the appearance of the standard ArcCAD dialog boxes as well as any customized dialog boxes from your application or other applications.

The file *ddarc.dcl* contains the definitions of all the function panels used by the standard release of ArcCAD. You can modify this file if you need to customize the appearance of the standard function panels.

Two corresponding LISP files are included in the \ARCAD\SOURCE directory: *pickers.lsp* and *ddarc.lsp*. These files supply the intelligence to the dialog boxes and are included as examples only. You should never modify the functionality in *pickers.lsp*. You may, however, modify *ddarc.lsp* to change the functionality in the standard function panels. If you modify *ddarc.lsp*, you must load this file only after ArcCAD is successfully loaded. ESRI does not support modifications made to any of the above files.

The following section describes the ArcCAD GUI functions.

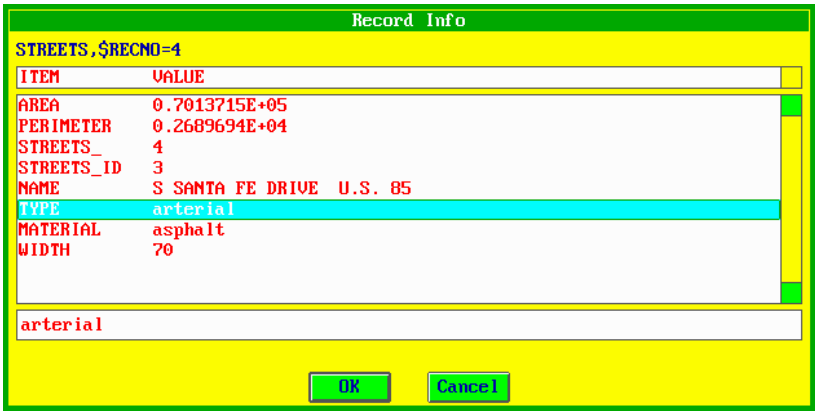
(arcd_browse1 *theme irec iopt*)

This function invokes the Record Info dialog box, which allows you to view or edit a specific record. The arguments *theme*, *irec* and *iopt* are used to control the initial appearance and behavior of the dialog box. The following figure shows an example of the Record Info dialog box.

The code fragment

```
(arcd_browse1 "streets" 4 2)
```

would display the Record Info dialog box as shown below:



STREETS,$RECNO=4

```
ITEM         VALUE
AREA         0.7013715E+05
PERIMETER    0.2689694E+04
STREETS_     4
STREETS_ID   3
NAME         S SANTA FE DRIVE  U.S. 85
TYPE         arterial
MATERIAL     asphalt
WIDTH        70
```

arterial

theme: A string specifying the name of the theme for which a specific record will be displayed. The specified theme must exist in the current drawing and must have an associated dBASE file. Please note that you cannot use *annotation* themes with this function.

irec: An integer specifying the record number of the dBASE file that will be displayed. If the record number does not exist, a warning message will be displayed indicating that the number entered is out of range.

iopt: This determines whether the dialog box is invoked to list the contents of the record number or to allow the item values to be modified for that record. If *iopt* is 1, the dialog box only lists the contents of the record and does not allow editing. The OK button is used to dismiss the dialog box and `T` is returned to the calling routine. If *iopt* is 2, the dialog box lists the contents of the record and allows editing of the item values. The OK button is used to accept changes made to that record's item values and `T` is returned to the calling routine. The Cancel button will abandon edits made to the item values and return a `nil` to the calling routine.

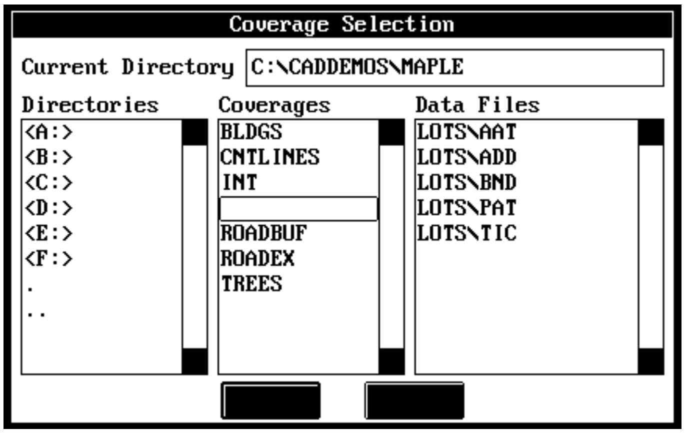
(arcd_cover *path iscover lonly*)

This function invokes a coverage or data file selection dialog box, which allows the user to pick either a coverage or a data file from a specified subdirectory. The arguments *path*, *iscover* and *lonly* are used to control the initial appearance and behavior of the dialog box. The following figure shows an example of the this dialog box.

The code fragment

```
(arcd_cover nil t nil)
```

would display the coverage selection dialog box as shown below:



Coverage Selection

Current Directory C:\CADDEMOS\MAPLE

Directories	Coverages	Data Files
<A:>	BLDGS	LOTS\AAT
<B:>	CNTLINES	LOTS\ADD
<C:>	INT	LOTS\BND
<D:>		LOTS\PAT
<E:>	ROADBUF	LOTS\TIC
<F:>	ROADEX	
.	TREES	
..		

path: The *path* is a string representing a pathname to a subdirectory to search. A drive letter is permitted in the path and you can use the forward slash instead of the backslash (but remember that you must use \\ to obtain one backslash in a string). If *path* evaluates to `nil` or `'.'`, the directory of the current drawing is assumed; otherwise, the pathname must be a full pathname (complete with disk drive letter and colon) to the directory to search.

iscover: The *iscover* argument determines whether a coverage selection or data file selection dialog box is invoked. If *iscover* evaluates to `T`, the coverage selection dialog box is displayed. If *iscover* evaluates to `nil`, the data file selection dialog box is displayed. Note that although both dialog boxes look identical, only the appropriate list box is activated for the selection. For example, if *iscover* evaluates to `nil`, the data file selection dialog box is invoked, and the OK button is activated only when you select one of the data files from the data files list box.

lonly: The *lonly* argument determines whether the dialog box is invoked to list the contents of subdirectories or to allow selection of a coverage or a data file name through the appropriate list box. If the dialog box is invoked only to list the contents of subdirectories, then the OK button dismisses the dialog box and `nil` is returned to the calling routine. If *lonly* evaluates to `T` the dialog box is used only for listing the contents of subdirectories. If *lonly* evaluates to `nil`, the OK button is activated on the selection of one of the items in the appropriate list box (based on the *iscover* parameter). After selecting a coverage or data file, double-click on the selection or press the OK button to dismiss the dialog box and return the full pathname of the selection to the calling routine.

(arcd_defthm *validtypes theme defvals*)

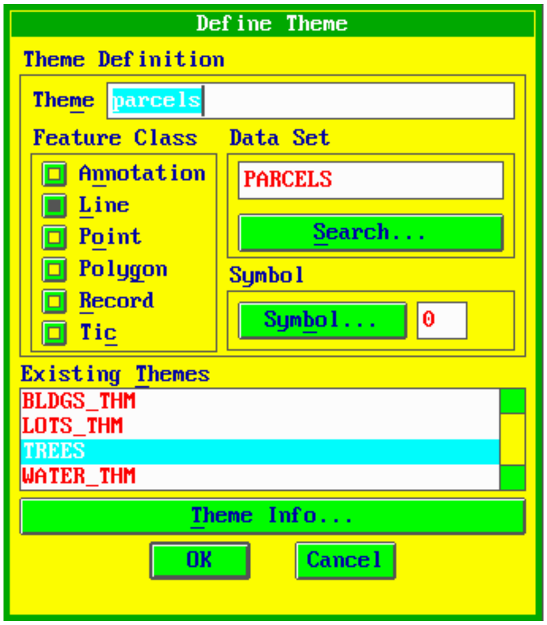
This function invokes a Define Theme dialog box which allows you to define a theme. You can control the feature class of themes that may be defined by setting the *validtypes* argument with an appropriate bit value. The *theme* argument is the name of the theme to define. The following figure shows an example of the theme definition dialog box.

The code fragment

```
(arcd_defthm (+ 1 2 4 8 16 32 64 128) “parcels” nil)
```

would display the Define Theme dialog box as shown below:

Note that the sum of the eight possible bit values for the *validtypes* argument instructs the **arcd_defthm** function to allow all feature classes to be defined.



Theme: parcels

☐	Annotation
■	Line
☐	Point
☐	Polygon
☐	Record
☐	Tic

Data Set: PARCELS

Symbol: 0

Existing Themes:

BLDGS_THM
LOTS_THM
TREES
WATER_THM

validtypes: The *validtypes* argument is a bit-coded integer that controls which feature classes may be defined within the Define Theme dialog box. *validtypes* may contain the following values:

Bit value	Meaning
1	*line* theme
2	*polygon* theme
4	*point* theme
8	*tic* theme
16	*annotation* theme
32	*dBASE record* theme
64	*SQL record* theme
128	*image* theme

The bit values can be added in any combination to enable the user to define a variety of theme types. For example, if you wish to be able to define *line* and *polygon* themes, set *validtypes* to 3.

defvals: This argument is used internally by ArcCAD and should always be set to `nil`.

After entering the appropriate information in the dialog box, press the OK button to dismiss the dialog box, define the theme, and return the theme name to the calling routine. If the theme is unable to be defined, `nil` is returned. The Cancel button dismisses the dialog box and returns `nil` to the calling routine.

Note: Future releases of ArcCAD might use additional **arcd_defthm** control bits, so avoid setting bits in your applications that aren't shown in the above table.

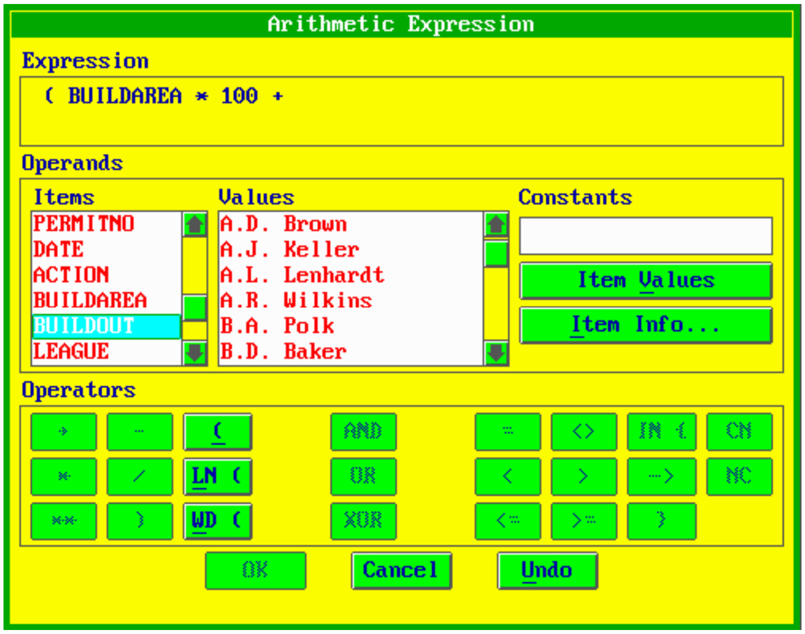
(arcd_equat *theme arith*)

This function invokes an Arithmetic or Logical Expression dialog box, which enables you to interactively build arithmetic or logical expressions on a specified theme. This dialog box is used to build expressions for calculating (modifying) item values or selecting theme features using logical expressions based on feature attributes. The following figure shows an example of the Arithmetic Expression dialog box.

The code fragment

```
(arcd_equat “parcels” t)
```

would display the Arithmetic Expression dialog box as shown below:



Expression: (BUILDAREA * 100 +

Items	Values
PERMITNO	A.D. Brown
DATE	A.J. Keller
ACTION	A.L. Lenhardt
BUILDAREA	A.R. Wilkins
BUILDOUT	B.A. Polk
LEAGUE	B.D. Baker

theme: The *theme* is the name of the theme for which you want to construct a logical or arithmetic expression. The specified theme must exist in the current drawing and must contain an associated data file. Please note that you cannot use annotation or image themes with this function.

arith: The *arith* argument determines the appearance of either the Arithmetic Expression or the Logical Expression dialog box. If *arith* evaluates to `T`, the Arithmetic Expression dialog box displays. If *arith* evaluates to `nil`, the Logical Expression dialog box displays.

After building the valid expression, the OK button dismisses the dialog box and returns the expression in the form of a string to the calling routine. The Cancel button dismisses the dialog box and returns `nil` to the calling routine.

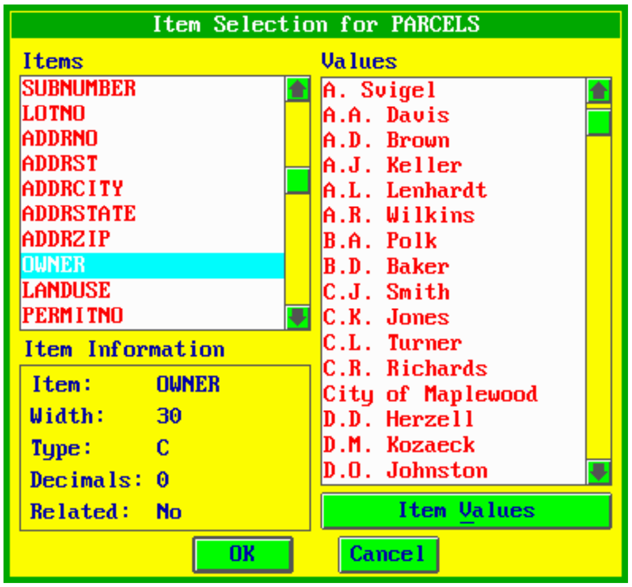
(arcd_item *theme validtypes lonly*)

This function invokes an Item Selection dialog box which allows you to pick an existing item name or list available items in the specified theme and controls which types of items to display using the *validtypes* argument. In addition, you can use the dialog box to list available items in the specified theme or actually pick one of the items for selection.

The code fragment

```
(arcd_item "parcels" (+ 1 2 4 8 16 32) nil)
```

would display the Item Selection dialog box with all of the items in the PARCELS theme as shown below:



Item Selection for PARCELS

Items

SUBNUMBER
LOTNO
ADDRNO
ADDRST
ADDRCITY
ADDRSTATE
ADDRZIP
OWNER
LANDUSE
PERMITNO

Item Information

Item:	OWNER
Width:	30
Type:	C
Decimals:	0
Related:	No

Values

A. Svigel
A.A. Davis
A.D. Brown
A.J. Keller
A.L. Lenhardt
A.R. Wilkins
B.A. Polk
B.D. Baker
C.J. Smith
C.K. Jones
C.L. Turner
C.R. Richards
City of Maplewood
D.D. Herzell
D.M. Kozaeck
D.O. Johnston

theme: The *theme* argument specifies the theme for which items are to be listed. If *theme* contains items, a list of those names is displayed in the items list box section of the dialog box. If the theme does not contain any items, an alert box is displayed.

validtypes: The *validtypes* argument controls which item types display in the dialog box. Four item types are currently available: *Character, Date*, *Integer* and *Numeric*. In addition, other item types available in dBASE, but not fully supported by ArcCAD, include *Logical* (logical data type) and *Memo* (MEMO field).

The *validtypes* argument is a bit-coded integer with values as shown below:

Bit value	Meaning
1	List *numeric* items
2	List *integer* items
4	List *character* items
8	List *date* items
16	List *logical* items
32	List *memo* fields

The bit values can be added in any combination to return a variety of item types. For example, if you wish to list *numeric* and *character* items, you can set the bit value to 5.

lonly: The *lonly* argument determines whether the dialog box is invoked to list the existing items or to allow selection of an item through the item list box. If the dialog box is invoked only to list the existing items in the specified theme, then the OK button is used to dismiss the dialog box and `nil` is returned to the calling routine. If *lonly* evaluates to `T`, the OK button dismisses the dialog box. If *lonly* evaluates to `nil`, the OK button and the Cancel button are enabled, allowing the user to select an item or dismiss the dialog box. After selecting an item name, press the OK button to dismiss the dialog box and return the item name to the calling routine.

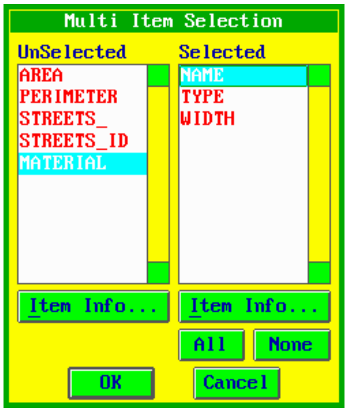
(arcd_items *theme validtypes dlist*)

This function invokes the Multi Item Selection dialog box, which allows you to view or select multiple items for a particular theme. The argument *dlist* controls which items of a theme display in the dialog box.

The code fragment

```
(arcd_items “streets” 7 nil)
```

would display the Multi Item Selection dialog box as shown below.



theme: A string specifying the name of the theme for which items are to be listed. If *theme* contains items, a list of those items displays in the UnSelected list box section of the dialog box.

validtypes: The *validtypes* argument controls which item types will be displayed in the Multi Item Selection dialog box. Four item types are currently available: *Character*, *Date*, *Integer* and *Numeric*. In addition to these four types, other item types available in dBASE, but not fully supported by ArcCAD, include *Logical* (logical data type) and *Memo* (MEMO field).

The *validtypes* argument is a bit-coded integer with values as shown below:

Bit value	Meaning
1	List *numeric* items
2	List *integer* items
4	List *character* items
8	List *date* items
16	List *logical* items
32	List *memo* fields

The bit values can be added in any combination to return a variety of item types. For example, to list *numeric* and *character* items, set the bit value to 5.

dlist: Puts item names into the *selected* list box of the browser. When **arcd_items** is invoked, all the valid items display in the unselected list box. Passing a list of item names through the argument *dlist* displays the item names of the list in the selected list box. The list of item names is compared to the item names retrieved by the *validtypes* argument. If the item in the list is present in the theme and is a valid type, then that item will be shown in the selected side of the dialog box. An item name not present in the theme or not a valid type will be ignored.

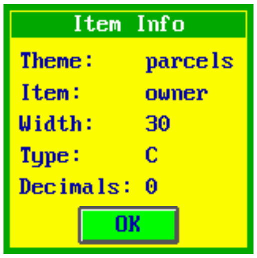
(arcd_iteminfo *theme item*)

This function displays an alert box which lists the definition information of the specified item. If *item* exists in *theme*, the theme name, the item name, and the item's type, width and number of decimal places displays. If *item* does not exist in the specified theme, the alert box is displayed with the message 'Item not found!'. The following figure shows an example of the alert box.

The code fragment

```
(arcd_iteminfo "parcels" "owner")
```

displays the Item Information alert box with its item named OWNER and its definition as shown below:


Item Info

```
Theme:     parcels
Item:      owner
Width:     30
Type:      C
Decimals:  0
```

theme: The *theme* is a valid theme name in the current drawing used to list one of its item names. The theme name must exist in the current drawing and must contain an associated data file. Please note that you cannot specify an annotation or image theme with this function.

item: A valid item name in the specified *theme*. If the specified item does not exist, the message 'Item not found.' is displayed in an alert box.

Note: This function always returns `T` to the calling routine.

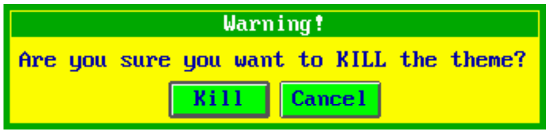
(arcd_msg *title accept reject msg*)

This function invokes an alert box with a user-specified error or warning message passed as the *msg* argument. The *title* argument allows you to specify the title of the alert box. The *accept* and *reject* arguments allow you to title the OK and Cancel buttons. The following figure shows an example of the alert box.

The code fragment

```
(arcd_msg "Warning!" "Kill" "Cancel" (list "Are you
                sure you want to KILL the theme?"))
```

would display the alert box as shown below:


Warning!

Are you sure you want to KILL the theme?

Notes

■ You must supply the message in the form of a list as shown in the above example.

■ If both the *accept* and *reject* arguments evaluate to `nil` the *accept* argument is activated to enable the OK button.

title: The optional *title* argument sets the title of the alert box. If the *title* evaluates to `nil`, no title appears in the alert box. The maximum length of the title depends upon and varies according to the display platform device.

accept: The *accept* argument sets the name of the *accept* (OK) button. If *accept* evaluates to `nil`, no *accept* button appears in the alert box. Selecting the *accept* button always returns `T` to the calling routine.

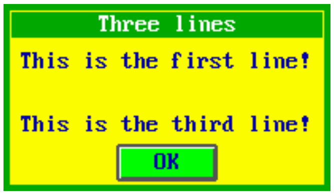
reject: The *reject* argument sets the name of the *reject* (Cancel) button. If *reject* evaluates to `nil`, no *reject* button displays in the alert box. Selecting the *reject* button always returns `nil` to the calling routine. Please note that if both the *accept* and *reject* arguments evaluate to `nil`, the *accept* argument activates and the button is named OK.

msg: The *msg* argument sets the error or warning message to be displayed. The message is passed in the form of a list that may contain a variable number of strings. If the list contains more than one string, each string appears on a separate line. An empty string (““) within a list may be used as a space between lines as shown below:

The code fragment

```
(arcd_msg “Three lines” “OK” nil
  (list “This is the first line!” ““ “This is the
  third line!”))
```

would display the alert box as shown below:


Three lines

This is the first line!

This is the third line!

Note that the line length and the number of lines in the alert box are device dependent, and any string too long to fit inside an alert box will be truncated.

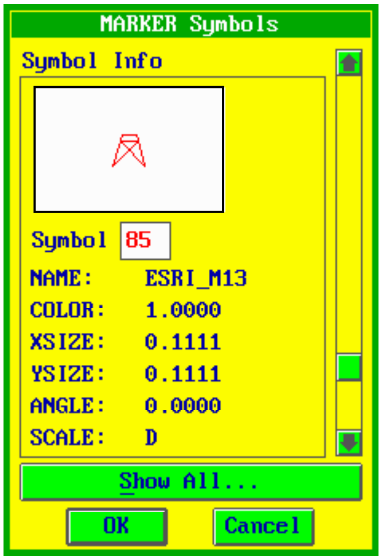
(arcd_symbol *type dsym lonly*)

This function invokes one of the four Symbol Selection dialog boxes. The *type* argument determines the type of symbol to display: Line, Marker, Shade or Text symbol. The *dsym* argument determines the current symbol that is displayed. The following shows an example of the Marker Symbols dialog box.

The code fragment

```
(arcd_symbol “Marker” 85 nil)
```

would display the Marker Symbols dialog box as shown below:



Symbol 85
NAME: ESRI_M13
COLOR: 1.0000
XSIZE: 0.1111
YSIZE: 0.1111
ANGLE: 0.0000
SCALE: D

type: The *type* argument controls the type of symbol to display in the dialog box. The symbols displayed are from the currently loaded symbol sets. ArcCAD supports four groups of symbols: marker, line, shade and text.

dsym: The *dsym* argument is used by **arcd_symbol** to determine which symbol to display first. Valid symbol numbers range between 0 and 100. This feature is especially useful for invoking the Symbol Selection dialog box with the symbol number that is being stored as part of a theme's definition.

Consider the following example, assuming that a theme named PARCELS exists in your drawing with the default symbol number 12:

```
...
...
(setq themelist (thmget “parcels”))
(if (null themelist)
    (princ “Theme not defined\n”)
    (setq sym (cdr (assoc 60 themelist)))
)
(arcd_symbol “Shade” sym nil)
...
...
```

The Shade Symbols dialog box is invoked with the current symbol number set to the theme's default symbol number. Choose the default symbol number or select another symbol using the Symbol Selection dialog box.

lonly: The *lonly* argument determines whether the dialog box is invoked to display the existing symbols or to allow selection of a symbol. If the dialog box is invoked only to display the symbols of a certain type, then the OK button is used to dismiss the dialog box and `nil` is returned to the calling routine. If *lonly* evaluates to `T`, the OK button dismisses the dialog box. If *lonly* evaluates to `nil`, the OK button and the Cancel button are enabled, allowing you to select a symbol or dismiss the dialog box. After selecting a symbol, press the OK button to dismiss the dialog box and return the symbol number to the calling routine.

(arcd_theme *validtypes etype define lonly theme*)

This function displays a Theme Selection dialog box from which to pick an existing theme or to list available themes in the current drawing. You can control which types of themes to display, using the *validtypes* parameter, and whether or not the data set must exist for the selected themes, using the *etype* argument. In addition, the optional *define* argument controls whether the 'Define Theme...' button, from which you can access the Define Theme dialog box, is enabled. The *theme* argument specifies the default theme. If this argument is blank, the current theme is used. The following figure shows an example of the Theme Selection dialog box.

The code fragment

```
(arcd_theme (+ 1 2 4 8 16 32) 0 t nil “parcels”)
```

would display the Theme Selection dialog box as shown below:

validtypes: The *validtypes* argument is used by **arcd_theme** to list only desired themes with the specified feature class. The *validtypes* is an integer (bit-coded) with values as shown below:

Bit value	Meaning
1	List *line* themes
2	List *polygon* themes
4	List *point* themes
8	List *tic* themes
16	List *annotation* themes
32	List *dBASE record* themes
64	List *SQL record* themes
128	List *image* themes

The bit values can be added in any combination to list a variety of theme types at the time of invocation. For example, if you wish to list only *line* and *polygon* themes, set the value of *validtypes* to 3.

etype: The *etype* argument is used by **arcd_theme** to list themes based on their respective geographic information system (GIS) data sets. In other words, you can list themes for which data sets exist or do not exist, or you can ignore the existence of the data sets altogether. The *dataset* argument is an integer with values as shown below:

Value	Meaning
0	Ignore GIS data set existence
1	GIS data set must exist
2	GIS data set must not exist
3	GIS data set and FAT must exist

define: The *define* argument controls the enabling of the 'Define Theme...' button in the Theme Selection dialog box. Use this button to access the Define Theme dialog box from which to define a theme. This feature is especially useful at *output theme* prompts where you may pick a theme from the available list or define one from within another command. If *define* evaluates to `T`, the 'Define Theme...' button is enabled; if it is set to `nil`, the 'Define Theme...' button doesn't display.

Note: When you press the 'Define Theme...' button, the Define Theme dialog box is displayed, and the OK and Cancel buttons of the **arcd_defthm** function control the value returned to the Theme Selection dialog box. For example, if you invoke the Define Theme dialog box from the Theme Selection dialog box, after entering appropriate information, the OK button returns the theme name to the **arcd_theme** function and appears in the Theme Selection dialog box.

lonly: The *lonly* argument determines whether the dialog box is invoked to list existing themes or to allow selection of a theme through the theme list box. If the dialog box is invoked only to list the existing themes in the current drawing (i.e., when *lonly* evaluates to `T`), only the OK button displays to dismiss the dialog box and `nil` is returned to the calling routine. If *lonly* evaluates to `nil`, the OK button and the Cancel button are enabled, allowing you to select a theme or dismiss the dialog box. After selection of a theme name, press the OK button to dismiss the dialog box and return the theme name to the calling routine. The Cancel button dismisses the dialog box and returns `nil` to the calling routine.

Note that the display of the 'Define Theme...' button depends on the *define* argument.

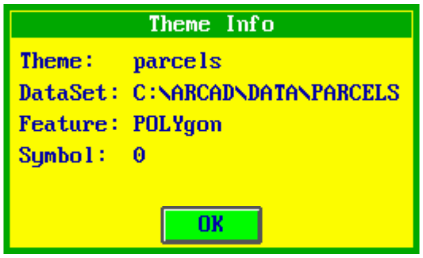
(arcd_thminfo *theme*)

This function invokes a Theme Information alert box which will list the definition information of the specified theme. If *theme* does not exist in the drawing, an alert box appears with the message ‘Theme name not found!’. The following figure shows an example of the Theme Information alert box.

The code fragment

```
(arcd_thminfo "parcels")
```

would display the Theme Information alert box as shown below:



```
Theme Info
Theme:    parcels
DataSet:  C:\ARCAD\DATA\PARCELS
Feature:  POLYgon
Symbol:   0
```

theme: The *theme* argument is a valid theme name and must exist in the current drawing. The theme’s data set, however, may or may not exist. If the data set does not exist in the specified theme, a ‘Data set does not exist.’ message appears in the theme information alert box.

Note: This function always returns `T` to the calling routine.

Chapter 7

Command interface

cmd function

The **cmd** function is a special function used to execute any valid ArcCAD® command from within AutoLISP®. This command is similar to the **command** function in standard AutoLISP. Refer to the *AutoCAD Customization Guide* for further details on the **command** function.

(cmd *list*)

This function executes ArcCAD commands from within AutoLISP and always returns `nil`. The *list* contains a set of arguments that represent an ArcCAD command and its parameters. It evaluates each argument and sends it to ArcCAD in response to successive prompts. The first atom in the *list* must be a valid ArcCAD command name.

The following is an example of the **cmd** function used to define a line theme named *roads*, with the geographic information system (GIS) data set *roads* and a symbol value *3*, using the **defthm** command.

```
(cmd (list “defthm” “roads” “Line” “roads” 3))
```

Each argument must be specified in the order it would be typed at the command prompt. Also, each argument should have the appropriate type, as specified in the *ArcCAD Command Reference*.

There are two special arguments available in the **cmd** function: *pause* and an empty string (“”). *pause* may be used to interrupt operation in order to request information from the user. Once the valid input is received, the function resumes.

Consider the following example:

```
(cmd (list  “joinitem” “roads” “roadsatt” “roads” “roads_id”
pause “”))
```

In the above example, while executing the ArcCAD **joinitem** command, the function pauses to let the user interactively type the *start item* option. When a valid input is found, the function resumes and accepts the default *link* option as it finds an empty string (“”) next to the *pause* argument, and completes the execution.

General rules in using the cmd function

■ Command names and options should be passed in as strings. Points as lists of two reals, and integers and reals should be passed as is. Note that keywords should be passed in with correct case (i.e., keyword matching is case sensitive).

■ A null string (“”) is equivalent to entering <CR> at the keyboard.

■ Use *pause* to prompt for interactive user input.

■ Commands executed from the **cmd** function are never echoed to the screen.

■ Commands with repeating (looped) dialog may not be used inside the **cmd** function due to the argument-passing restrictions of AutoLISP. ArcCAD commands with repeating dialog must only be provided with one iteration of that dialog’s arguments, including a null string (“”) to terminate that iteration.

Index

ArcCAD® Extensions to ADS™

™

Contents

Chapter 1

Theme access

A theme is a collection of geographic phenomena or an organizing principle that is used to link AutoCAD® entities to geographic information system (GIS) data sets. Each theme has a unique name, a feature class, a pointer referencing a corresponding GIS data set (a PC ARC/INFO-compatible coverage or a database data file), and a symbol number. Themes are stored as point entities in the current drawing and maintained on a special layer named ESRI_THEMES. The parameters of the theme definition are maintained in the extended entity data of the point entities that are created on this special layer. This layer is not visible to the user and should always be in a *frozen* state. You must never attempt to edit the contents of this layer. Doing so can destroy the theme definitions and therefore corrupt all the links that are maintained between AutoCAD entities and ArcCAD® features.

The theme manipulation functions described in the following section maintain and manipulate the extended entity data and the links for you. These functions are used to manage themes in your drawing and maintain the links between the ArcCAD features and the corresponding AutoCAD entities. For example, using the **arc_thmdef** function, you can define a theme to create or display GIS data sets. You can use the **arc_thmdel** function to delete a theme and the links to the GIS data set. Similarly, using the **arc_thmexi** function, you can check the validity of a theme before performing any operation on the theme. The **arc_thmlst** function lists the available themes in your drawing, and the **arc_thmget** function lets you retrieve the contents of the theme definition in the form of a list, which can be used to modify the definition of the theme using the **arc_thmmod** function.

The following are some of the theme limitations that are applied to theme access functions:

- The maximum length of a theme name is 31 characters.

- The maximum number of themes that can be defined and linked to GIS data sets in a single AutoCAD drawing is 511.

- You cannot define more than one theme with the same feature class and GIS data set, with the exception of *record* themes.

Theme manipulation functions

The following functions can be used to create, modify, list and delete some of the parameters that define a theme.

arc_thmdef

Input	Output
-1 dummy ename	`RTSTR` theme
1 theme name	
2 Feature class	
3 GIS data set	
57 User_ID increment value	
58 Next available User_ID #	
60 Symbol	
63 Features present	

Xloads required: arcad

This function creates a new theme in the drawing. If *theme* is successfully created, the name of the theme is returned as a `RTSTR`. If *theme* cannot be created, `RTNIL` is returned. There are seven possible feature classes available to the user: Annotation, Line, Point, Polygon, Record (dBASE), Image, and Tic.

Note: SQL record themes are not supported by the ArcCAD ADS™ extensions.

This function does not perform any user interaction. All of the parameters must be supplied. If user interaction is desired, use the **arc_cmd** function to invoke the user command that defines themes.

Example

```
/* this code fragment creates a line theme named
'ROADS' that uses a GIS data set named
'DEMO\ROADS' and a line symbol of 2.
*/

long ename[2];
char theme[32];
struct resbuf  *in_rb, *out_rb;

ename[0]=0;
ename[1]=0;
```

```
in_rb  =  ads_buildlist  (
     RTSTR, “arc_thmdef”,
     -1, ename,
     1, “roads”,
     2, “Line”,
     3, “demo\\roads”,
     60, 2,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype  ==  RTSTR)    {
     strcpy(theme,out_rb->resval.rstring);
     ads_printf(“theme defined\n”);
}
else ads_printf(“theme not defined\n”);
```

Note: If it happens to be the first theme created in your drawing, the software first creates a frozen, invisible layer, ESRI_THEMES.

arc_thmdel

Input	**Output**
`RTSTR` theme	`RTSTR` theme
`RTSHORT` option	

Xloads required: arcad

This function removes a theme from the drawing. The actual output of this function is dependent upon the value of *option* provided. This function will either return the theme name as a `RTSTR` or `RTNIL`.

If *option* evaluates to 0, only the links associated with *theme* will be removed. *Theme* will remain defined in the drawing. If the links are successfully removed from the drawing, the name of the theme will be returned as a `RTSTR`. If the links cannot be removed from the drawing, `RTNIL` will be returned.

If *option* evaluates to 1, the associated links *and* the theme definition will be removed from the drawing. If *theme* is successfully deleted, the name of the theme will be returned as a `RTSTR`. If *theme* cannot be deleted, `RTNIL` will be returned.

When a theme is deleted from the drawing, the associated links are also removed from all entities belonging to the theme. In addition, all feature selection sets derived from the theme are removed.

Example

```
struct resbuf  *in_rb, *out_rb;
char theme[32];

/* This code fragment removes only the links
associated with a theme named ‘sample’.
*/
in_rb  =  ads_buildlist  (
     RTSTR, “arc_thmdel”, RTSTR, “sample”,
     RTSHORT, 0,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype  ==  RTSTR)    {
     strcpy(theme,out_rb->resval.rstring);
     ads_printf(“removed links\n”);
}
else ads_printf(“links not removed\n”);

/* This code fragment removes the associated links and
the definition of the theme named “roads”.
*/

in_rb  =  ads_buildlist  (
     RTSTR, “arc_thmdel”, RTSTR, “roads”,
     RTSHORT, 1,
     NULL);
ads_invoke (in_rb, out_rb)
if (out_rb->restype  ==  RTSTR)    {
     strcpy(theme,out_rb->resval.rstring);
     ads_printf(“theme and links removed\n”);
}
else ads_printf(“theme and links not removed\n”);
```

Note: If you delete a GIS data set reference by a theme, it is always a good idea to use the **arc_thmdel** function to remove the links associated with that theme. The ArcCAD command **kill** handles this automatically.

arc_thmexi

Input	**Output**
`RTSTR` theme	`RTSTR` theme

Xloads required: arcad

This function checks for the existence of a theme in the drawing. If *theme* exists, the name of the theme will be returned as a `RTSTR`. If *theme* does not exist, `RTNIL` will be returned.

Example

```
struct resbuf *in_rb,*out_rb;
char theme[32];

/* This code fragment checks for the existence of a
theme named ‘sample’.
*/
in_rb  =  ads_buildlist  (
     RTSTR, “arc_thmexi”, RTSTR, “sample”,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype  ==  RTSTR)    {
     strcpy(theme,out_rb->resval.rstring);
     ads_printf(“found it\n”);
}
else ads_printf(“does not exist\n”);
```

arc_thmget

Input	**Output**
`RTSTR` theme	-1 dummy ename
	1 theme name
	2 class (read only)
	3 GIS data set (read only)
	57 User_ID increment value
	58 Next available User_ID#
	60 symbol
	63 Features present *(read only)*

Xloads required: arcad

This function returns the definition of an existing theme. If *theme* exists, a `RESBUF` will be returned that contains the theme’s name, class, GIS data set, and symbol number. If *theme* does not exist in the drawing, `RTNIL` will be returned.

Example

```
/* This code fragment retrieves the definition of a
theme named 'roads'.
*/

struct resbuf *in_rb, *out_rb;
char name[32], class[32], source[132];
int symbol;

in_rb = ads_buildlist (
     RTSTR, "arc_thmget", RTSTR, "roads",
     NULL);
ads_invoke (in_rb,out_rb);
while (out_rb != NULL) {
     if (out_rb->restype == 1)
          strcpy(name,out_rb->resval.rstring);
     if (out_rb->restype == 2)
          strcpy(class,out_rb->resval.rstring);
     if (out_rb->restype == 3)
          strcpy(source,out_rb->resval.rstring);
     if (out_rb->restype == 60)
          symbol=out_rb->resval.rint;
     if (out_rb->restype != RTNIL)
          out_rb=out_rb->rbnext;
     else out_rb=NULL;
}
```

arc_thmlst

Input	**Output**
`RTSHORT` class	`RTSTR` theme name1
`RTSHORT` data set	`RTSTR` theme name2
	.....
	.....
	`RTSTR` theme nameN

Xloads required: arcad

This function returns a list of the themes defined in the current drawing. If the drawing contains any themes, a `RESBUF` containing the theme names is returned. If the drawing does not contain any themes, `RTNIL` is returned. The optional *class* and *dataset* arguments let you list themes in a variety of combinations. Please note that you cannot omit only one of the arguments. You must either omit both the *class* and *dataset* arguments or supply both of them.

class: This optional argument determines the feature class of themes returned. The *class* argument is an integer (bit-coded) with values as shown below:

Bit value	Meaning
1	List *line* themes
2	List *polygon* themes
4	List *point* themes
8	List *tic* themes
16	List *annotation* themes
32	List dBASE *record* themes
64	List SQL *record* themes
128	List *image* themes

The bit values can be added in any combination to return a variety of theme feature classes. For example, if you wish to list all *line* and *polygon* themes, set the value of *class* to 3.

Caution: Future versions of ArcCAD might use additional **arc_thmlst** control bits, so avoid setting bits in your applications that aren't in the above table.

dataset: This optional argument instructs **arc_thmlst** to return themes based on the existence of their respective GIS data sets. In other words, you can retrieve themes for which a data set exists or does not exist, or you can ignore the existence of the data set altogether. The *dataset* argument is an integer with values as shown below:

Value	Meaning
0	Ignore GIS data set existence
1	GIS data set must exist
2	GIS data set must not exist

Data set existence rules

The GIS data set exists when the following conditions are met for *line*, *point*, *polygon*, *tic* and *annotation* themes:

- The GIS data set (coverage) has been created by adding one or more features to the theme.
- A theme is defined with its GIS data set referencing an existing coverage in the specified subdirectory.
- For a *record* theme, the GIS data set exists if it contains at least one item definition. It is not dependent on the number of records in the file.

Example

```
/* This code fragment returns a list of all line and
polygon themes in your current drawing.
*/

struct resbuf *in_rb, *out_rb, *cur_rb;

in_rb = ads_buildlist (
     RTSTR, "arc_thmlst", RTSHORT, 3, RTSHORT, 0,
     NULL);
ads_invoke (in_rb, &out_rb);
cur_rb = out_rb;
while (cur_rb->restype == RTSTR && cur_rb != NULL) {
     ads_printf (“Theme name is: %s\n”,
           cur_rb->resval.rtstring);
     cur_rb = cur_rb->rbnext;
}
ads_relrb (in_rb);
ads_relrb (out_rb);
```

arc_thmmod

Input	**Output**
RTSTR theme	RTSTR theme
-1 dummy ename	
1 new theme name	
2 class *(read-only)*	
3 GIS data set *(read-only)*	
57 User_ID increment value	
58 Next available User_ID#	
60 symbol	
63 Features present *(read only)*	

Xloads required: arcad

This function modifies the definition of an existing theme. If *theme* is successfully modified, the name of the theme associated with the restype of 1, in the input `RESBUF`, is returned as a `RTSTR`.

If *theme* cannot be modified, `RTNIL` will be returned. The DXF group codes in the *buildlist* arguments must be in the same format as received from the **arc_thmget** function as shown in the example below. *Please note that the class and GIS data set values are read-only.*

Example

```
/* This code fragment changes the name of a previously
defined theme named ‘sample’ to ‘roads’.
*/
struct resbuf *in_rb, *out_rb, *new_rb;
char name[32], class[32], source[132];
int symbol;
long ename[2];

/* get existing values */

in_rb = ads_buildlist (
     RTSTR, “arc_thmget”, RTSTR, “sample”,
     NULL);
ads_invoke (in_rb,out_rb)
while (out_rb != NULL) {
     if (out_rb->restype == 1)
          strcpy(name,out_rb->resval.rstring);
     if (out_rb->restype == 2)
          strcpy(class,out_rb->resval.rstring);
     if (out_rb->restype == 3)
          strcpy(source,out_rb->resval.rstring);
     if (out_rb->restype == 60)
          symbol=out_rb->resval.rint;
     if (out_rb->restype != RTNIL)
          out_rb=out_rb->rbnext;
     else out_rb=NULL;
}

/* create new resbuf with modified values */

ename[0]=0;
ename[1]=0;
new_rb=ads_buildlist(
     RTSTR, “arc_thmmod”, -1, ename,
     1, “roads”, 2, class, 3, source, 60, symbol,
     NULL);
ads_invoke (new_rb, out_rb);
```

```
if (out_rb->restype  ==  RTSTR)    {
     strcpy(name,out_rb->resval.rstring);
     if (strcmp(name,“roads”)==0)
          ads_printf(“modification complete\n”);
     else ads_printf(“modification failed\n”);
}
else ads_printf(“modification failed\n”);
```

Chapter 2

Feature access

A comprehensive set of AutoLISP® and ADS™ functions, provided by Autodesk®, Inc., allows you to access AutoCAD® entities. Running parallel to these functions, ArcCAD® software supports similar functions that access features in ArcCAD databases. Using these functions, you can select features, retrieve their values and modify them. You can use AutoLISP variables to hold selection sets derived in this fashion so that you can manipulate selection sets of features.

Feature selection set manipulation

In order to manipulate features, the user must be able to indicate which objects are to be considered. This process involves the selection of features based upon combinations of spatial, graphical and attribute criteria specified as a series of logical expressions. This series of functions allows the user to perform and maintain selection sets, and retrieve information concerning these objects of interest.

There are two special ArcCAD data types implemented to provide access to ArcCAD features: a *feature name* and a *feature selection set*. A feature name is a pointer into the ArcCAD database from which AutoLISP can find the appropriate feature and its corresponding attributes (if any exist). A feature selection set is simply a collection of feature names.

Feature selection sets

A feature selection set in ArcCAD is a set of features that are grouped together based on a series of spatial, logical and arithmetic expressions. At the command level, there is one feature selection set maintained for each theme (except *annotation*) defined in the drawing. Initially, all features belonging to a database are selected for that theme's selection set. Using the **reselect**, **aselect**, **nselect** and **clearsel** commands, the user can manipulate the feature selection sets to display desired features and their corresponding attribute values (if any exist). The last operation on a theme's feature selection set is always stored as the current feature selection set for that theme.

At the AutoLISP and ADS level, the user can create and manipulate feature selection sets using a set of feature selection functions (explained in later sections). By using these functions, the user can achieve the same results as one who uses ArcCAD commands for query and display purposes. There are, however, obvious advantages to using these functions to customize the software at the AutoLISP and ADS level.

As explained earlier, at the command level, there is always one and only one feature selection set maintained for any given theme. By using feature selection functions, the user can store multiple feature selection sets for a given theme and use any one of these feature selection sets to replace the current feature selection set for that same theme.

While making multiple feature selection sets using AutoLISP and ADS functions, the user has options to select features from the current feature selection set or from the entire geographic information system (GIS) data set associated with that theme. For example, if the user creates a new feature selection set from the theme’s selection set, the function behaves exactly like a **reselect** command. On the other hand, if the user specifies the *all* option while deriving a feature selection set (refer to the **arc_fssget** function), the user has the ability to select a subset of features from the entire GIS data set of a given theme.

The functions **arc_fssand**, **arc_fssor** and **arc_fssxor** are also available to the user to create new feature selection sets by combining two valid feature selection sets based on Boolean AND, OR, XOR combinations.

Features may be selected spatially or by using a series of logical expressions based on valid feature attributes in the corresponding feature attribute table for a given theme, as explained below:

Spatial selection

Spatial selection sets can be performed using the following modes:

- Circle crossing
- Circle within
- Window crossing
- Window within
- Polygon crossing
- Polygon within
- Previous (latest feature selection set)
- Interactive (by specifying the appropriate mode)

For further details on spatial selection, refer to the **arc_fssget** function.

Logical expressions

Logical expressions in ArcCAD have three components: operands, logical operators and logical connectors.

Logical operands

- The name of an item in a data file (e.g., `STREAMS_ID`)
- A constant numerical value (e.g., `10`)
- A character string in single quotation marks (e.g., `'HIGH'`)
- An internal variable (e.g., `$RECNO`)

Logical operators

`EQ` or =	Operand-1 is equal to Operand-2
`NE` or <>	Operand-1 is not equal to Operand-2.
`GE` or >=	Operand-1 is greater than or equal to Operand-2.
`LE` or <=	Operand-1 is less than or equal to Operand-2.
`GT` or >	Operand-1 is greater than Operand-2.
`LT` or <	Operand-1 is less than Operand-2.
`CN`	Operand-1 contains the character expression Operand-2. Used with character operands only (e.g., `NAME CN 'MAIN'`).
`NC`	Operand-1 does not contain the character expression Operand-2. Used with character operands only (e.g., `NAME NC 'MAIN'`).
`IN`	Operand-1 is contained in the set of numeric constants of character strings specified in Operand-2. This set of constants or character strings must be enclosed in { } braces. The elements in the set must be separated by commas, unless they are being used to express a range, in which case, -> is used to separate the elements forming the lower- and upper-inclusive limits of the range. A range defined between two character strings is based on the ASCII number sequence, which is alphabetical. No blank spaces should separate any of the elements within the brackets.

Note: Computer roundoff can alter the values of real numbers. This can cause a problem when specifying real numbers in a [logical expression] that require equality. When using expressions of equality, the operands must match exactly for the match to be found. For example, the value .01139 does not equal .0114. In such cases, use an expression that includes a range of real values (i.e., "HEIGHT GT .01139 AND HEIGHT LT .01141").

Logical connectors

`AND`	For the whole expression to be evaluated as true, the logical expressions on both sides of the `AND` must be true.
`OR`	For the whole expression to be evaluated as true, the logical expression on one or the other side of the `OR` must be true. The whole expression will also be evaluated as true if both logical expressions are true.
`XOR`	For the whole expression to be evaluated as true, the logical expression on one and only one side of the `XOR` must be true. If both logical expressions are true or both are false, the condition will be evaluated as false.

The simplest logical expressions take the following form:

[operand-1] [logical-operator] [operand-2]

For example, the following string can be used as a simple logical expression:

```
"CLASS LT 8"
```

Up to eight logical expressions of this simple form can be combined to form more complex expressions by using the logical connectors. For example, the following string can be used as a complex logical expression:

```
"CLASS GE 2 AND CLASS LT 8 OR SUIT = 'MODERATE'"
```

Note: If the item value is of type character, the value must be enclosed in single quotes (example: SUIT = 'MODERATE') and is case sensitive.

There is no limit to the number of [operand-1] [logical-operator] [operand-2] combinations and logical connectors that can be used in a single expression. However, the entire expression must be less than 254 characters in length.

All logical operators and connectors have equal precedence. The expression is evaluated from left to right. However, parentheses can be used to change the default order of evaluation such that expressions within parentheses are performed first. Operations inside the innermost set of parentheses have the highest precedence.

Each element of a logical expression (i.e., operand, logical operator, logical connector, parenthesis) must be separated by blanks, except when using the IN operator.

Arithmetic expressions

Arithmetic expressions in ArcCAD have the following components:

Numeric operands

• An item name
• A constant (e.g., `10`)
• An internal variable (e.g., `$RECNO`)

Arithmetic operators

+	Addition
-	Subtraction
/	Division
*****	Multiplication
******	Exponentiation
LN	Logarithm Calculates the natural logarithm of the operand it precedes. The operand must be a positive number.
WD	Width computation Calculates the width in characters of the operand it precedes excluding trailing blanks. The operand must be a character item or a literal string.

Arithmetic operators have the following precedence from highest to lowest:

1) LN, WD
2) **
3) *, /
4) +, -

Operands of equal precedence are performed as they are encountered moving from left to right through the expression. Parentheses can be used to override inherent precedence. Operations within the innermost set of parentheses are performed first.

Note: There is no unary minus operator for negating an operand in ArcCAD. For example, the expression -AGE evokes an error message (instead, specify -1 * AGE). Also, all arithmetic operations in ArcCAD are performed in double precision. As a result, an expression involving integer operands may be evaluated as having a fractional part.

Examples of arithmetic expressions:

```
SUIT = ( SOIL + 2 * TERRAIN ) / 12

LAB_WIDTH = ( WD ( LABEL ) + 4 ) * 0.22
```

Display width limitations

A number assigned to a numeric type item with 0 decimal places that exceeds the defined width for its item definition is displayed as asterisks (***). The item value is lost.

The values of a numeric type item defined with 1 or more decimal places that exceeds the defined width for that item will be displayed in scientific notation.

Internal variables

ArcCAD provides you with three internal variables that can be used in logical and arithmetic expressions.

$RECNO—the record number of a record in the selected data file.

$PI—the value for *pi* (3.14159...), which is the ratio of a circle's circumference to its diameter.

$E—the value for *e* (approximately 2.71828), which is the base of the number system for natural logarithms.

These internal variables can be used as operands anywhere within a logical or arithmetic expression; for example

```
"$RECNO GT 100"
```

This selection expression will find all records from the selection set whose record number is greater than 100.

Feature selection set manipulation functions

The following functions are used to create, modify, list and delete feature selection sets.

arc_fssadd

Input	**Output**
`RTSTR` featname	`RTSHORT` fss
`RTSHORT` fss	

Xloads required: arcad

This function adds a named feature to a feature selection set. The actual output of this function is dependent upon the value of *fss* provided. This function will either return a feature selection set as a `RTSHORT` or return `RTNIL`. When a new member is added to *fss*, it is added to all feature selection sets bound to *fss*. In other words, if *fss* is assigned to other variables, they also reflect the addition.

If *fss* does not evaluate to `RTNIL`, this function will add *featname* to *fss*. If *featname* is successfully added to the feature selection set, the new feature selection set is returned as a `RTSHORT`. If the *featname* is already a member of *fss*, the operation will be ignored and the original feature selection set will be returned as a `RTSHORT`. If *featname* and *fss* are not derived from the same theme, `RTNIL` will be returned.

If *fss* evaluates to `RTNIL`, a new feature selection set will be created. This new feature selection set will contain *featname* as its only member.

Example

```
/* This code fragment adds the last feature in the
theme ‘ROADS’ to an existing feature selection set
named fss1
*/
char f1[32];
int  fs;
struct resbuf  *in_rb,*out_rb;

/* Get name of last feature in theme */

in_rb =   ads_buildlist(
     RTSTR, “arc_featlast”, RTSTR, “roads”,
     NULL);
ads_invoke(in_rb,out_rb);
if (out_rb->restype==RTSTR) {
     strcpy(f1,out_rb->resval.rstring);
     /* Add feature  to selection set */
     in_rb=ads_buildlist(
          RTSTR, “arc_fssadd”, RTSTR, f1,
          RTSHORT, fss1,
          NULL);
     ads_invoke(in_rb,out_rb);
     if (out_rb->restype==RTSHORT) {
          fs=out_rb->resval.rint;
          if (fs==fss1) ads_printf(“success\n”);
          else ads_printf(“failure\n”);
     }
     else ads_printf(“failure\n”);
}
else ads_printf(“failure\n”);
```

arc_fssand

Input	**Output**
`RTSHORT` fss1	`RTSHORT` fss
`RTSHORT` fss2	

Xloads required: arcad

This function creates a new feature selection set, which contains all of the feature names in both of the feature selection sets provided. If both *fss1* and *fss2* are derived from the same theme, a new feature selection set will be returned as a `RTSHORT`. If the feature selection sets are not derived from the same theme, `RTNIL` will be returned.

If the two feature selection sets do not have any members in common, the resulting feature selection set will not have any members. This function is equivalent to a Boolean `AND` of the two feature selection sets.

Example

```
/* This code fragment creates a new feature selection
set that contains all members that are common to both
feature selection sets fss1 and fss2.
*/

int  fss3;
```

```
struct resbuf   *in_rb, *out_rb;

in_rb = ads_buildlist  (
     RTSTR,  “arc_fssand”, RTSHORT,  fss1,
     RTSHORT,  fss2,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype == RTSHORT)    {
     fss3=out_rb->resval.rint;
     ads_printf(“success\n”);
}
else ads_printf(“failure\n”);
```

arc_fssclr

Input	**Output**
`RTSHORT` fss	`RTSHORT` fss

Xloads required: arcad

This function removes all of the members from a feature selection set. If all members are successfully removed from *fss*, the feature selection set is returned as a `RTSHORT`. If all the members cannot be removed, `RTNIL` will be returned.

Example

```
/* This code fragment removes all members from the
feature selection set fss1
*/

int  fss;
struct resbuf   *in_rb, *out_rb;

in_rb  =  ads_buildlist  (
     RTSTR, “arc_fssclr”, RTSHORT, fss1,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype==RTSHORT) {
     fss=out_rb->resval.rint;
     if (fss==fss1) ads_printf(“success\n”);
     else ads_printf(“failure\n”);
}
else ads_printf(“failure\n”);
```

arc_fssdel

Input	**Output**
`RTSTR` featname	`RTSHORT` fss
`RTSHORT` fss	

Xloads required: arcad

This function removes a named feature from a feature selection set. If *featname* is successfully removed from *fss*, the feature selection set *fss* will be returned as a `RTSHORT`. If *featname* cannot be removed from the feature selection set, `RTNIL` will be returned.

Example

```
/* This code fragment removes the last feature in the
theme 'ROADS' from the existing feature selection set
fss1.  The feature selection set is checked to see if
the feature is a member prior to its removal.
*/
char feat1[32];
int  fss;
struct resbuf  *in_rb, *out_rb;

in_rb  =  ads_buildlist  (
     RTSTR,  "arc_featlast", RTSTR,  "roads",
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype  ==  RTSTR)    {
     strcpy(feat1,out_rb->resval.rstring);
     /* check for membership in fss */
     in_rb  =  ads_buildlist  (
          RTSTR,  "arc_fssmemb",
          RTSTR, feat1, RTSTR,  "roads",
          NULL);
     ads_invoke(in_rb,out_rb);
     if (out_rb->restype==RTSTR) {
          /* remove feature name from fss */
          in_rb = ads_buildlist  (
               RTSTR, "arc_fssdel",
               RTSTR, feat1,
               RTSHORT, fss1,
               NULL);
          ads_invoke (in_rb, out_rb);
          if (out_rb->restype == RTSHORT    {
               fss = out_rb->resval.rint;
               if (fss==fss1)
```

```
                        ads_printf(“success\n”);
                   else ads_printf(“failure\n”);
              }
              else ads_printf(“failure\n”);
         }
         else ads_printf(“Not a member of fss\n”);
    }
    else ads_printf(“failure\n”);
```

arc_fssfree

Input	**Output**
`RTSHORT` fss	`RTNIL`

Xloads required: arcad

This function frees resources allocated to a feature selection set *(fss)*. This allows other feature selection sets to be created since a finite number of feature selection sets (128) are available to the user. This function will always return `RTNIL`.

Example

```
/* This code fragment frees resources allocated to the
feature selection set fss1.
*/

struct resbuf   *in_rb, *out_rb;
in_rb  =  ads_buildlist  (
     RTSTR, “arc_fssfree”,
     RTSHORT, fss1,
     NULL);
ads_invoke (in_rb, out_rb);
/* always returns RTNIL */
```

arc_fssget

Input	**Output**
`RTSTR` theme	`RTSHORT` fss
`RTSTR` state	
mode	
option1	
option2	

The following chart shows the relationships between the values of *mode*, *option1*, and *option2*. All of these modes are of type `RTSTR`, with the exception of `RTNIL`.

Description	Modes	Option1	Option2
Circle (crossing and within)	CC	RTPOINT	RTREAL
Circle (within)	CW	RTPOINT	RTREAL
Expression	E	RTSTR	RTNIL
Previous	P	RTNIL	RTNIL
Polygon (crossing and within)	PC	RTLB RTPOINT RTPOINT ... RTLE	RTNIL
Polygon (within)	PW	RTLB RTPOINT RTPOINT ... RTLE	RTNIL
Window (crossing and within)	WC	RTPOINT	RTPOINT
Window (within)	WW	RTPOINT	RTPOINT
Interactive	RTNIL	RTNIL	RTNIL

Xloads required: arcad

This function creates a feature selection set based upon the *state* of the selection and a selection method indicated by the user. This function will only process features that are members of a given *theme*. All other features will be ignored.

state: The argument *state* represents the current state of the selection set. By default, all the features of a given theme are selected. If the user wants to select a subset of features from a current selection set, the *state* of the selection would be "*C*" (current). If the user wants to select features from the entire database associated with the specified theme, the *state* of the selection would be "*A*" (all). In other words, if the *state* argument is "*C*", this function behaves similarly to the **reselect** command. On the other hand, if the *state* argument is "*A*", this function behaves similarly to the **aselect** command.

mode: There are currently nine different selection modes available to the user. A list of these modes is provided below.

"CC" *pt1 radius*	Selects all features crossing or inside a circle whose center is *pt1* (`RTPOINT`) and the radius is *radius* (`RTREAL`).
"CW" *pt1 radius*	Selects all features inside a circle whose center is *pt1* (`RTPOINT`) and the radius is *radius* (`RTREAL`).
"E" *expression nil*	Selects all features that satisfy a logical expression. *expression* (`RTSTR`) contains one or more valid logical expressions. For further details, refer to the section on logical expressions in this chapter.
"P" *nil nil*	Returns the currently selected features of the theme. This option will always return the current selection for the theme and assumes that the *state* argument is always "*C*" (current). If the user specifies option "*A*" (all) as the *state* argument, the function ignores this option and assumes option "*C*" as the *state* argument. However, since the *state* argument is always evaluated by the function, the user should not skip this option or specify any other option keyword except "*C*" or "*A*".
"PC" *list1 nil*	Selects all features crossing or inside a polygon whose vertices are stored in a *list1*. The format of the list is `RTLB`, `RTPOINT`, `RTPOINT`, ... , `RTLE`.
"PW" *list1 nil*	Selects all features inside a polygon whose vertices are stored in a *list1*. The format of the list is `RTLB`, `RTPOINT`, `RTPOINT`, ... , `RTLE`.
"WC" *pt1 pt2*	Selects all features crossing or inside a box whose opposite corners are *pt1* (`RTPOINT`) and *pt2* (`RTPOINT`).
"WW" *pt1 pt2*	Selects all features inside a box whose opposite corners are *pt1* (`RTPOINT`) and *pt2* (`RTPOINT`).
nil nil nil	Selects all features interactively. Only one mode of input will be performed. When this option is used, the values of *mode, option1,* and *option2* are all `RTNIL`.

Example

```
/* This code fragment shows several examples of the
fssget function using the theme ‘LANDUSE’
*/

ads_point pt1,pt2;
int  fss1,fss2,fss3;
struct resbuf  *in_rb, *out_rb;

/* This section selects all features that cross or are
within a user defined window */

ads_getpoint(NULL,“First corner: ”,pt1);
ads_getcorner(pt1,“opposite corner: ”,pt2);
in_rb= ads_buildlist(
     RTSTR, “arc_fssget”, RTSTR, “landuse”, RTSTR,
     “a”,RTSTR, “wc”, RTPOINT, pt1,
     RTPOINT,pt2,NULL);
ads_invoke(in_rb,out_rb);
if (out_rb->restype==RTSHORT) {
     fss1= out_rb->resval.rint;
}
else {
     ads_printf(“fss1 failed\n”);
}

/* This section selects features using a logical
expression
*/

in_rb= ads_buildlist(
     RTSTR, “arc_fssget”, RTSTR, “landuse”, RTSTR,
     “a”,RTSTR, “e”, RTSTR, “lucode = 2”, RTNIL,
     NULL);
ads_invoke(in_rb,out_rb);
if (out_rb->restype==RTSHORT) {
     fss2= out_rb->resval.rint;
}
else {
     ads_printf(“fss2 failed\n”);
}

/* This section selects features interactively
*/

in_rb= ads_buildlist(
     RTSTR, “arc_fssget”, RTSTR, “landuse”, RTSTR,
     “a”,RTNIL, RTNIL, RTNIL,
     NULL);
ads_invoke(in_rb,out_rb);
if (out_rb->restype==RTSHORT) {
     fss3= out_rb->resval.rint;
}
else ads_printf("fss3 failed\n");
```

Warning: The results of each **arc_fssget** operation are always stored as the theme's feature selection set. When this function is used, the base of selectable features is dependent on the value of the *state* argument. For instance, if you specify the *state* argument as "C", the base of selectable features is the theme's feature selection set. On the other hand, if you specify "A" as the *state* argument, the base of selectable features is all features in the theme. Therefore, the value of the *state* argument influences the results of the selection and should be used with care.

arc_fsslength

Input	**Output**
`RTSHORT` fss	`RTREAL` count

Xloads required: arcad

This function returns the number of members contained in a feature selection set. If the specified feature selection set is valid and contains zero or more members, the function returns a nonnegative real number as a `RTREAL`. If the specified feature selection set is not valid, `RTNIL` will be returned.

Example

```
/* This code fragment returns the number of members in
a user defined feature selection set */

char theme[32];
int  fss1;
ads_real  length;
struct resbuf  *in_rb, *out_rb;

/* Get theme name from user */

ads_getstring(0,"Theme name: ",theme);

/* Get user defined feature selection set */

in_rb  =  ads_buildlist  (
     RTSTR, “arc_fssget”, RTSTR, theme,
     RTSTR, “a”, RTNIL, RTNIL, RTNIL,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype == RTSHORT)    {
     fss1 = out_rb->resval.rint;
     in_rb  =  ads_buildlist  (
          RTSTR, “arc_fsslength”,
          RTSHORT, fss1,
          NULL);
     ads_invoke (in_rb, out_rb);
     if (out_rb->restype == RTREAL)    {
          length = out_rb->resval.rreal;
          ads_printf(“there are %7.0f members\n”,
               length);
     }
     else ads_printf(“failure\n”);
}
else ads_printf(“failure\n”);
```

arc_fssmemb

Input	**Output**
`RTSTR` f1	`RTSTR` status
`RTSHORT` fss	

Xloads required: arcad

This function determines if a named feature is a member of a feature selection set *fss*. If *featname* is member of *fss*, `RTSTR` will be returned. If *featname* is not a member of *fss*, `RTNIL` will be returned.

Example

```
/* This code fragment determines if a feature feat1 is
a member of the feature selection set fss1 */

struct resbuf   *in_rb, *out_rb;
in_rb  =  ads_buildlist  (
     RTSTR, “arc_fssmemb”,
     RTSRT, feat1, RTSHORT, fss1,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype==RTSTR)
     ads_printf(“feature is a member\n”);
else ads_printf(“feature is not a member\n”);
```

arc_fssname

Input	**Output**
`RTSHORT` fss	`RTSTR` featname
`RTREAL` index	

Xloads required: arcad

This function returns the name of the indexed feature in the feature selection set. The *index* must always be specified as `RTREAL`. The index of the first member of the feature selection set is 0.0. If *fss* contains a member at the *index* position, the feature name will be returned as a `RTSTR`. If *index* is negative or greater than the number of members contained in the feature selection set, `RTNIL` will be returned.

Example

```
/* This code fragment retrieves the name of the 6th
member of the feature selection set fss1.
*/

struct resbuf   *in_rb, *out_rb;
char feat1[32];

in_rb  =  ads_buildlist  (
     RTSTR,  “arc_fssname”, RTSHORT, fss1,
     RTREAL,  5.0,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype  ==  RTSTR)
     strcpy( feat1,out_rb->resval.rstring);
else ads_printf(“feature does not have 6
     members\n”);
```

arc_fssnot

Input	**Output**
`RTSHORT` fss	`RTSHORT` fss

Xloads required: arcad

This function switches the members of a feature selection set. All of the current members of the feature selection set are replaced by all of the members that were removed from the feature selection set. This function is equivalent to a Boolean complement of the feature selection set. If members of a feature selection set are successfully complemented, the feature selection set will be returned as a

`RTSHORT`. If the feature selection set cannot be complemented, `RTNIL` will be returned.

Example

```
/* This code fragment creates a feature selection set
that contains all features of the theme ‘ROADS’ that
are not selected by the user
*/

struct resbuf   *in_rb, *out_rb;
int  fss,fss1;

/* Get user’s feature selection set */

in_rb  =  ads_buildlist  (
     RTSTR,  “arc_fssget”, RTSTR,  “roads”,
     RTSTR,  “a”, RTNIL, RTNIL, RTNIL,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype  ==  RTSHORT)    {
     fss1 = out_rb->resval.rint;
     /* complement feature selection set */
     in_rb  =  ads_buildlist  (
          RTSTR,  “arc_fssnot”,
          RTSHORT,  fss1,
          NULL);
     ads_invoke (in_rb, out_rb);
     if (out_rb->restype  ==  RTSHORT)    {
          fss = out_rb->resval.rint;
          if (fss==fss1) ads_printf(“success\n”);
          else ads_printf(“failure\n”);
     }
     else ads_printf(“failure\n”);
}
else ads_printf(“failure\n”);
```

arc_fssor

Input	**Output**
`RTSHORT` fss1	`RTSHORT` fss
`RTSHORT` fss2	

Xloads required: arcad

This function creates a new feature selection set that contains all of the members of both of the feature selection sets provided. If both *fss1* and *fss2* are derived from the same theme, a new feature selection set will be returned as a `RTSHORT` that contains members in both *fss1* and *fss2*. If the feature selection sets are not derived from the same theme, `RTNIL` will be returned. This function is equivalent to a Boolean `OR` of the two feature selection sets.

Example

```
/* This code fragment creates a feature selection set
that contains a combination of features from two
different feature selection sets.  This example uses
the theme ‘LANDUSE’ and the user defined item ‘LUCODE’
*/

struct resbuf   *in_rb, *out_rb;
int  fss1, fss2, fss3;

/* Get user’s first feature selection set */

in_rb  =  ads_buildlist  (
     RTSTR,  “arc_fssget”, RTSTR,  “landuse”,
     RTSTR,  “a”, RTNIL, RTNIL, RTNIL,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype  ==  RTSHORT)
     fss1 = out_rb->resval.rint;
else {
     ads_printf(“failure\n”);
     fss1= -1;
}

/* Get user’s second feature selection set */

in_rb  =  ads_buildlist  (
     RTSTR,  “arc_fssget”, RTSTR,  “landuse”,
     RTSTR,  “a”, RTSTR, “e”, RTSTR, “lucode = 2”,
     RTNIL,
     NULL);
```

```
ads_invoke (in_rb, out_rb);
if (out_rb->restype  ==  RTSHORT)
     fss2 = out_rb->resval.rint;
else {
     ads_printf(“failure\n”);
     fss2= -1;
}

/* Combine both feature selection sets */
if (fss1 > -1 && fss2 > -1) {
     in_rb  =  ads_buildlist  (
          RTSTR,  “arc_fssor”, RTSHORT, fss1,
          RTSHORT, fss2,
          NULL);
     ads_invoke (in_rb, out_rb);
     if (out_rb->restype  ==  RTSHORT)
          fss3  =  out_rb->resval.rint;
     else ads_printf(“failure\n”);
}
```

arc_fssput

Input	**Output**
`RTSHORT` fss	`RTSHORT` fss

Xloads required: arcad

This function replaces a theme’s feature selection set with the members of the feature selection set provided. If the replacement is successful, the name of the feature selection set will be returned as a `RTSHORT`. If the replacement cannot be performed, `RTNIL` will be returned.

Example

```
/* This code fragment creates several feature
selection sets and overwrites the theme’s selection
set with the very first feature selection set
generated.  This example uses the theme ‘ROADS’.
*/

struct resbuf   *in_rb, *out_rb;
int  fss1, fss2, fss3;
```

```
/* Get user’s first feature selection set. Remember
that the theme’s selection set will be the same as the
feature selection set generated.
*/

in_rb  =  ads_buildlist  (
     RTSTR,  “arc_fssget”, RTSTR,  “roads”
     RTSTR,  “a”, RTNIL, RTNIL, RTNIL,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype  ==  RTSHORT)
     fss1 = out_rb->resval.rint;
else {
      ads_printf(“failure\n”);
     fss1= -1;
}

/* Get user’s second feature selection set. Remember
that the theme’s selection set will not be previously
generated, but will be generated by this function call
*/

in_rb  =  ads_buildlist  (
     RTSTR,  “arc_fssget”, RTSTR,  “roads”,
     RTSTR,  “a”, RTNIL, RTNIL, RTNIL,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype  ==  RTSHORT)
     fss2 = out_rb->resval.rint;
else {
     ads_printf(“failure\n”);
     fss2= -1;
}

/* Overwrite theme’s selection set with that of the
first feature selection set generated
*/

if (fss1 > -1 && fss2 > -1) {
     in_rb  =  ads_buildlist  (
          RTSTR,  “arc_fssput”, RTSHORT, fss1,
          NULL);
     ads_invoke (in_rb, out_rb);
     if (out_rb->restype  ==  RTSHORT) {
          fss3  =  out_rb->resval.rint;
          if (fss3 == fss1) ads_printf(“success\n”);
          else ads_printf(“failure\n”);
     }
     else ads_printf(“failure\n”);
}
```

arc_fssxor

Input	**Output**
`RTSHORT` fss1	`RTSHORT` fss
`RTSHORT` fss2	

Xloads required: arcad

This function creates a new feature selection set, which contains all of the members that are not in both of the sets provided. If both *fss1* and *fss2* are derived from the same theme, a new feature selection set will be returned as a `RTSHORT`. If the feature selection sets are not derived from the same theme, `RTNIL` will be returned. This function is equivalent to a Boolean EXCLUSIVE OR of the two feature selection sets. If *fss1* is identically equal to *fss2*, an empty feature selection set will be created.

Example

```
/* This code fragment creates a feature selection set
that contains all features that are not in both
feature selection sets fss1 and fss2.
*/

int  fss3;
struct resbuf  *in_rb, *out_rb;

in_rb  =  ads_buildlist  (
     RTSTR,  “arc_fssxor”, RTSHORT,  fss1,
     RTSHORT,  fss2,
     NULL);
ads_invoke  (in_rb,  out_rb);
if (out_rb->restype  ==  RTSHORT)
     fss3  =  out_rb->resval.rint;
else ads_printf(“failure\n”);
```

Feature name functions

The following functions are used to retrieve feature names and entity names.

arc_entfeat

Input	**Output**
`RTENAME` ename	`RTSTR` featname1
`option`	`RTSTR` featname2
	`.....`
	`RTSTR` featnameN

The following table shows the relationship between the value of *option* and the output of the function.

Description	**Option value**
Return all feature names	RTNIL
Return a feature name corresponding to a theme	RTSTR theme

Xloads required: arcad

This function returns the corresponding feature name(s) of features linked to the named entity. The *ename* must be the name of the main entity. This function searches *ename* and retrieves a list of the appropriate feature names that correspond to the entity. This function always returns the name of the main feature. You should use the **arc_featnext** function to access the subfeatures of the main feature (if any exist). The actual output of this function is dependent upon the value of *theme*. This function will either return a `RESBUF` containing feature names or `RTNIL`.

If *theme* does not evaluate to `RTNIL`, this function will check to see if *ename* is a member of *theme*. If the entity is a member of the theme, a `RESBUF` containing only the corresponding feature name will be returned. If the entity is not a member of the theme, `RTNIL` will be returned.

If *theme* evaluates to `RTNIL`, a `RESBUF` containing all of the corresponding feature names is returned. If the entity is not a member of any theme, `RTNIL` will be returned.

Example

```
/* This code fragment returns a list of themes that
contains the feature feat1 as a member.
*/

struct resbuf   *in_rb, *out_rb;
char theme[32];
int  ok;

ok=1;
in_rb  =  ads_buildlist  (
     RTSTR,  “arc_entfeat”, RTENAME,  e1,
     RTNIL,
     NULL);
ads_invoke  (in_rb,  out_rb);
while (ok) {
     if (out_rb->restype==RTSTR) {
          strcpy(theme,out_rb->resval.rstring);
          ads_printf(“%s\n”,theme);
          out_rb=out_rb->rbnext;
          if (out_rb==NULL) ok=0;
     }
     else ok=0;
}
```

Notes

■ In a polygon theme, links are established only to the label point(s) and, therefore, you must select the label point in order to retrieve the corresponding polygon feature. Selecting the polygon boundary always returns `nil`. You must use the **arc_featnext** function to access the line (polygon boundary) subfeatures.

■ The *tic* and *record* themes do not maintain links in ArcCAD and, therefore, the **arc_entfeat** function on these themes will always return `nil`.

arc_featent

Input	**Output**
`RTSTR` featname	`RTENAME` ename1

Xloads required: arcad

This function returns the corresponding entity name of a feature. If *featname* is rendered in the drawing, the corresponding entity name will be returned as a

`RTENAME`. If *featname* has not been rendered in the drawing, `RTNIL` will be returned.

Example

```
/* This code fragment retrieves the corresponding
entity name for the feature feat1.
*/

struct resbuf   *in_rb, *out_rb;
long ent1[2];

in_rb  =  ads_buildlist  (
     RTSTR, “arc_featent”,
     RTSTR, feat1,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype==RTENAME)
     ent1[0]=out_rb->resval.rlname[0];
     ent1[1]=out_rb->resval.rlname[1];
}
else ads_printf(“failure\n”);
```

arc_featlast

Input	**Output**
`RTSTR` theme	`RTSTR` featname

Xloads required: arcad

This function returns the name of the last feature created in the specified theme. If *theme* contains features, the name of the last feature will be returned as a `RTSTR`. If *theme* does not contain any features, `RTNIL` will be returned.

This function is useful to retrieve the *featname* after the user has created a new feature via the **arc_cmd** function.

Example

```
/* This code fragment retrieves the name of the last
feature in the theme ‘ROADS’
*/

struct resbuf   *in_rb, *out_rb;
char feat1[32];
```

```
in_rb  =  ads_buildlist  (
     RTSTR, “arc_featlast”, RTSTR, “roads”,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype  ==  RTSTR)
     strcpy(feat1,out_rb->resval.rstring);
else ads_printf(“failure\n”);
```

arc_featnext

Input	**Output**
`RTSTR` theme	`RTSTR` featname
`RTSTR` featname	
`RTSHORT` level	

Xloads required: arcad

This function returns the name of the next feature in a theme. The actual feature name returned is a function of the values of *featname* and *level*.

If *featname* evaluates to `RTNIL`, the value of *level* is ignored. The resulting name returned will be that of the first feature defined in *theme*. If *theme* does not contain any features, `RTNIL` will be returned.

If *featname* does not evaluate to `RTNIL`, the resulting feature name returned will be dependent upon the current value of *featname* and *level*. If *theme* does not contain any features, `RTNIL` will be returned.

The following table shows the relationships between *featname*, *level*, and the resulting feature name returned by this function.

Current Feature Name	Level 1	Level 2	Level 3
Annotation	Next annotation	Next annotation	Next annotation
Line header	Next line header	Next vertex of line	Next vertex of line
Line vertex	Next line header	Next vertex of line or line end	Next vertex of line or line end
Line end	Next line header	Next line header	Next line header
Point	Next point	Next point	Next point
Polygon header	Next polygon	Next point or line header of polygon	Next point or line header of polygon
Polygon point	Next polygon	Next point or line header of polygon	Next point or line header of polygon
Polygon line header	Next polygon	Next line header of polygon	Next vertex of line
Polygon line vertex	Next polygon	Next line header of polygon	Next vertex of line or line end
Polygon line end	Next polygon	Next line header of polygon	Next line header of polygon
Polygon end	Next polygon	Next polygon	Next polygon
Record	Next record	Next record	Next record
Tic	Next tic	Next tic	Next tic

Example

```
/* This code fragment retrieves the name of the first
vertex in the third line subfeature of the first
polygon feature in the theme ‘LANDUSE’
*/

struct resbuf   *in_rb, *out_rb;
int  ok, cnt;
char feat[32], subfeat[32];

/* Get name of first polygon in theme */

in_rb=ads_buildlist (
   RTSTR, “arc_featnext”, RTSTR, “landuse”, RTNIL,
   RTNIL, NULL);
ads_invoke(in_rb,out_rb);
if (out_rb->restype == RTSTR) {
   ok=1;
   strcpy(feat1,out_rb->resval.rstring);
}
```

```
else {
  ok=0;
  ads_printf("no topology in theme\n");
}

/* Read until 3rd line is found */

cnt=0;
while (ok>0) {
  /* Get next subfeature which may be point or
  line
  */
  in_rb=ads_buildlist (
    RTSTR, "arc_featnext", RTSTR, "landuse",
    RTSTR, feat1, RTSHORT, 2, NULL);
  ads_invoke(in_rb,out_rb);
  if (out_rb->restype != RTSTR) {
    ads_printf("polygon has no subfeatures\n");
    ok=0;
  }
  else {
    strcpy(subfeat,out_rb->resval.rstring);
    /* Check feature to see if it is a line */
    in_rb=ads_buildlist (
      RTSTR, "arc_featget", RTSTR, subfeat,
      RTNIL, NULL);
    ads_invoke(in_rb,out_rb);
    while (out_rb != NULL) {
      if (out_rb->restype==0) {
        if (strcmp("LINE",out_rb-
>resval.rstring)==0) {
          cnt++;
          if (cnt==3) {
            ok=0;
            out_rb=NULL;
          }
          else out_rb=out_rb->rbnext;
        }
        else out_rb=out_rb->rbnext;
      }
      else out_rb=out_rb->rbnext;
    }
    strcpy(feat1,subfeat);
  }
}

/* Get first vertex of line feature */

if (cnt==3) {
  in_rb=ads_buildlist (
  RTSTR, “arc_featnext”, RTSTR, “landuse”,
  RTSTR, subfeat, RTSHORT, 3, NULL);
  ads_invoke(in_rb,out_rb);
  if (out_rb->restype == RTSTR)
    ads_printf(“success\n”);
  else
    ads_printf(“failure\n”);
}
```

arc_featthm

Input	**Output**
`RTSTR` featname	`RTSTR` theme

Xloads required: arcad

This function returns the name of the theme in which the named feature is a member. If *featname* is a valid feature name, the name of the theme that contains the named feature is returned as a `RTSTR`. If *featname* is not a valid feature, `RTNIL` will be returned.

Example

```
/* This code fragment retrieves the name of the theme
that the feature feat1 is a member
*/

struct resbuf  *in_rb, *out_rb;
char theme[32];

in_rb  =  ads_buildlist  (
     RTSTR, “arc_featthm”, RTSTR, feat1,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype  ==  RTSTR)
     strcpy(theme,out_rb->resval.rstring);
else ads_printf(“failure\n”);
```

Feature data functions

The following functions are used to create, modify, and delete features and feature data in a theme.

arc_featdel

Input	**Output**
`RTSTR` featname	`RTSTR` featname

Xloads required: arcad

This function removes the named feature from its GIS data set. If *featname* is successfully removed from the GIS data set, the name of the feature will be returned as a `RTSTR`. If *featname* cannot be removed from the GIS data set, `RTNIL` will be returned.

This function invalidates all feature selection sets that are bound to the feature name. In addition, the entity-feature link associated with the corresponding feature will be dropped (if any exists).

Example

```
/* This code fragment removes the feature feat1 from
its GIS data set
*/

struct resbuf   *in_rb, *out_rb;
char result[32];

in_rb  =  ads_buildlist  (
     RTSTR, “arc_featdel”, RTSTR, feat1,
     NULL);
ads_invoke (in_rb, out_rb);
if (out_rb->restype  ==  RTSTR)
     ads_printf(“feature removed\n”);
else ads_printf(“failure\n”);
```

arc_featget

Input	**Output**
RTSTR featname	DXF Restypes
itemlist	

The following table shows the possible values for *itemlist* and their meaning.

Meaning	**Itemlist value**
Ignore item values	RTNIL

Retrieve values for selected item names	RTLB
	  RTSTR
	  RTSTR
	  ...
	  RTSTR
	RTLE

Xloads required: arcad

This function retrieves the named feature's data from the theme. If *featname* exists, a RESBUF containing its data is returned. If *featname* does not exist, RTNIL will be returned. This function also returns the values of a list of items specified by *itemlist* (see the **arc_itmlst** function).

The format of RESBUF is dependent on the type of the feature class being retrieved. The format of each of the feature classes is shown below. The order of the sublists may vary and should not be assumed to be in the order shown.

There are seven possible feature classes: Annotation, Point, Tic, Line, Record, Polygon and Image. Note that Image and SQL record themes cannot be used with this function. The contents of RESBUF for all seven feature classes are described in the following sections:

■ ANNOTATION features are returned in the following format. This format is styled after the TEXT listing from the **ads_entget** function.

Restype	Resval	Description	Comments
-1	<Entity name: 0>	dummy ename	
0	“ANNOTATION”	feature class	Read only
1	text	text string	
6	featname	feature name	Read only
40	hgt	height	
41	wdtscl	width scale	
42	level	annotation level	
43	symbol	symbol number	
59	Internal-ID	Internal-ID	Read only
10	x1 y1	1st position point	
11	x2 y2	2nd position point	Optional
12	x3 y3	3rd position point	Optional
13	x4 y4	4th position point	Optional
14	x5 y5	1st arrow point	Not Used
15	x6 y6	2nd arrow point	Not Used
16	x7 y7	3rd arrow point	Not Used

■ POINT features are returned in the following format. This format is styled after the POINT listing from the **ads_entget** function.

Restype	Resval	Description	Comments
-1	<Entity name: 0>	dummy ename	
0	"POINT"	feature class	Read only
6	featname	feature name	Read only
10	x y	coordinates	
40	poly1	polygon id	Read only
58	User-ID	User-ID	
59	Internal-ID	Internal-ID	Read only

■ TIC features are returned in the following format. This format is styled after the POINT listing from the **ads_entget** function.

Restype	Resval	Description	Comments
-1	<Entity name: 0>	dummy ename	
0	"TIC"	feature class	Read only
6	featname	feature name	Read only
10	x y	coordinates	
58	User-ID	User-ID	
59	Internal-ID	Internal-ID	Read only

■ LINE features are returned in the following format. When the **arc_featget** function is called with the name of the line feature, the following list is returned.

Restype	Resval	Description	Comments
-1	<Entity name: 0>	dummy ename	
0	"LINE"	feature class	Read only
6	featname	feature name	Read only
70	vcnt	number of vertices	Read only
40	node1	from node	Read only
41	node2	to node	Read only
42	poly1	left polygon	Read only
43	poly2	right polygon	Read only
58	User-ID	User-ID	
59	Internal-ID	Internal-ID	Read only

When **arc_featget** is called with the name of a line *subfeature*, the following list is returned.

Restype	Resval	Description	Comments
-1	<Entity name: 0>	dummy ename	
0	"VERTEX"	subfeature class	Read only
6	featname	subfeature name	Read only
10	x y	coordinate	

When **arc_featget** is called with the name of the *last line subfeature*, the following list is returned to indicate the end of the feature.

Restype	Resval	Description	Comments
-1	<Entity name: 0>	dummy ename	
0	"LINEEND"	end of line marker	Read only

■ RECORD features are returned in the following format. When the **arc_featget** function is called, the following list is returned.

Restype	Resval	Description	Comments
-1	<Entity name: 0>	dummy ename	
0	"RECORD"	subfeature class	Read only
6	featname	subfeature name	Read only
59	Internal-ID	Internal-ID	Read only

To obtain item names and values, the *itemlist* option should be used.

■ POLYGON features are returned in the following format. When the **arc_featget** function is called with the name of the polygon feature, the following list is returned.

Restype	Resval	Description	Comments
-1	<Entity name: 0>	dummy ename	
0	"POLYGON"	feature class	Read only
6	featname	feature name	Read only
58	User-ID	User-ID	Not Used
59	Internal-ID	Internal-ID	Read only
70	pcnt	number of points	Read only
71	lcnt	number of lines	Read only
10	x y	coordinates of centroid	Read only
11	xmin ymin	box min coordinates	Read only
12	xmax ymax	box max coordinates	Read only

When processing a polygon feature, the points are always accessed before the lines and their vertices. When **arc_featget** is called with the name of a polygon subfeature, one of the following lists will be returned.

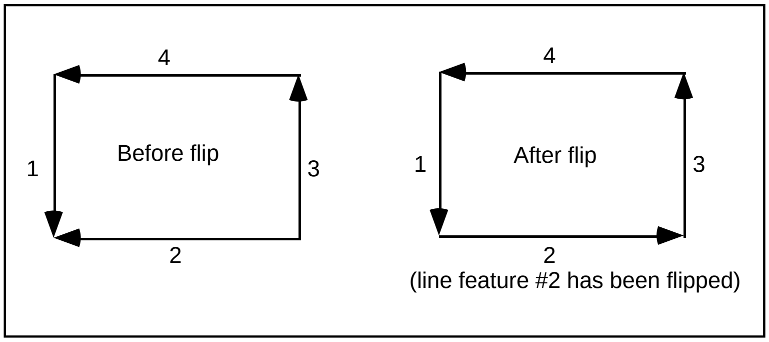
Restype	Resval
-1	<Entity name: 0>
0	“POINT”
6	featname
10	x y
40	poly1
58	User-ID
59	Internal-ID

Restype	Resval
-1	<Entity name: 0>
0	“LINE”
6	featname
70	vcnt
71	flip
72	island
40	node1
41	node2
42	poly1
43	poly2
58	User-ID
59	Internal-ID

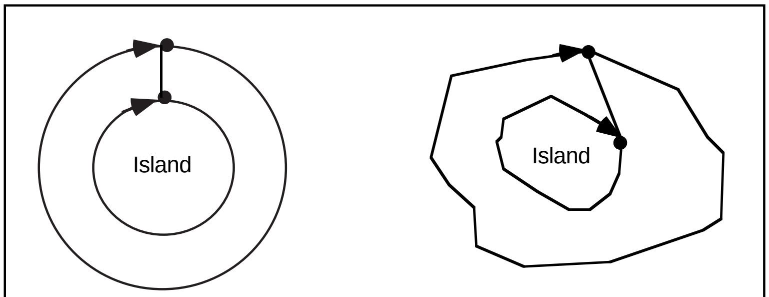
Please remember that polygon features have subfeatures called points and lines and sub-subfeatures called vertices. Also remember that the first polygon feature in a polygon theme is always the *universe polygon.* For additional details, refer to the *ArcCAD User’s Guide*.

Note: The two *restypes* **71** (flip) and **72** (island) mentioned in the line header of a polygon deserve further explanation:

`RESTYPE 71`: This code is optional and is used to indicate flip line coordinates. The value **1** indicates that the line's coordinates should be flipped, and the default value **0** represents *no flip*. The following figure demonstrates the usage of the **flip** option.



`RESTYPE 71`: This code is optional and is used to indicate that the remaining line (sub) features of the polygon are part of an island within the current polygon. The following figure demonstrates the potential use of the **island** option in a polygon theme:



Island formation in polygon themes

Restype	Resval
-1	<Entity name: 0>
0	“VERTEX”
10	x y
6	featname

Restype	Resval
-1	<Entity name: 0>
0	“LINEEND”

Restype	Resval	Description	Comments
-1	<Entity name: 0>	dummy ename	
0	“POLYEND”	end of polygon marker	Read only

Please remember that polygon features have subfeatures called points and lines. Line subfeatures have sub-subfeatures called vertices.

itemlist option

If the *itemlist* option does not evaluate to `RTNIL`, the **arc_featget** function retrieves both feature data and feature attribute values from the specified theme. The list of attribute data is appended to the end of the list normally returned with the main features. This new list will have the following format:

Restype	Resval	Description	Comments
-3		item flag	
1001	“ESRI”	application name	
1040 or 1000	real 1 or text 1	real value 1 or text value 1	Optional

The *optional* designation indicates that only one of the two possible group codes will be present. The storage type of the item determines which group code is output.

Items are only returned when *featname* is the name of a main feature. To illustrate, consider the following example where **arc_featget** is used with the *itemlist* option to retrieve the coordinates of a line feature and its corresponding attribute values. When the function is called with the name of the line feature with *itemlist* option, the following list is returned.

Restype	**Resval**
-1	<Entity name: 0>
0	“LINE”
6	featname
70	vcnt
40	node1
41	node2
42	poly1
43	poly2
58	User-ID
59	Internal-ID
-3	
1001	“ESRI”
1040 or 1000	real1 text1
1040 or 1000	real2 text2
…..	
1040 or 1000	realN textN

When the **arc_featnext** function is called with the name of the line feature, the name of the first subfeature (vertex) is returned. When **arc_featget** is called with the name of the first subfeature, the following list is returned.

Restype	Resval
-1	<Entity name: 0>
0	"VERTEX"
10	x y
6	featname

You will notice that the list returned from the function does not contain any reference to item data because the feature retrieved is not a main feature.

arc_featmake

Input	**Output**
`RTSTR` theme	`RTSTR` featname
featlist	
itemlist	

The following table shows the possible values for *itemlist* and their meaning.

Meaning	Itemlist value
Ignore item values	RTNIL

Retrieve values for selected item names	RTLB
	  RTSTR
	  RTSTR
	  ...
	  RTSTR
	RTLE

Xloads required: arcad

This function creates a new feature in a theme. If the feature is successfully created, the name of the feature will be returned. If the feature cannot be created, `RTNIL` will be returned.

If *itemlist* evaluates to `RTNIL`, the corresponding item values for the feature are ignored. If *itemlist* does not evaluate to `RTNIL`, the item values for the feature will be retrieved. Values will not be returned for item names that are not in *itemlist*. In addition, the values that are returned will be in the same order as their item names.

Featlist is assumed to be in the same format as that returned by the **arc_featget** function. The only exception to this is when polygon features are being created.

A polygon theme is composed of lines and points. The actual polygon features are not created until the theme's polygon topology has been processed. To create a polygon theme, add points and line features to the theme. Calls to the **arc_featlast** function will return `RTNIL` until the theme is processed as mentioned earlier. In addition, **arc_featmake** will return a dummy feature name.

When creating line features, multiple calls to **arc_featmake** will be required. Each of the subsequent calls will define the vertex subfeatures. When all of the vertex subfeatures have been defined, a call to **arc_featmake** with the following code fragment must be used to terminate the feature.

```
long ename[0]=0;
long ename[1]=0;
ads_buildlist ( RTSTR, “arc_featmake”,
     -1, ename, 0, “LINEEND”, NULL);
ads_invoke(in_rb,out_rb);
```

Example

```
/* This code fragment creates a line feature with four
vertices. The coordinates of the vertices are 0,0  1,1
2,0 and 3,1. Additionally, the User-ID of the line is
123. This example assumes that linetheme is a valid
line theme.
*/

struct resbuf   *in_rb, *out_rb;
long ename[2];
double    point[3];

/* main feature header */

ename[0]=0;
ename[1]=0;
in_rb= ads_buildlist ( RTSTR, “arc_featmake”,
     RTSTR, linetheme, -1, ename,
     0, “LINE”, 40, 0.0, 41, 0.0, 42, 0.0, 43, 0.0,
     58, 123.0, RTNIL, NULL);
ads_invoke(in_rb, out_rb);
/* first subfeature */
```

```
point[0]=0.0;
point[1]=0.0;
point[2]=0.0;
in_rb= ads_buildlist ( RTSTR, “arc_featmake”,
     RTSTR, linetheme, -1, ename,
     0, “VERTEX”, 10, point, RTNIL, NULL);
ads_invoke(in_rb, out_rb);

/* second subfeature */

point[0]=1.0;
point[1]=1.0;
point[2]=0.0;
in_rb= ads_buildlist ( RTSTR, “arc_featmake”,
     RTSTR, linetheme, -1, ename,
     0, “VERTEX”, 10, point, RTNIL, NULL);
ads_invoke(in_rb, out_rb);

/* third subfeature */

point[0]=2.0;
point[1]=0.0;
point[2]=0.0;
in_rb= ads_buildlist ( RTSTR, “arc_featmake”,
     RTSTR, linetheme, -1, ename,
     0, “VERTEX”, 10, point, RTNIL, NULL);
ads_invoke(in_rb, out_rb);

/* fourth subfeature */

point[0]=3.0;
point[1]=1.0;
point[2]=0.0;
in_rb= ads_buildlist ( RTSTR, “arc_featmake”,
     RTSTR, linetheme, -1, ename,
     0, “VERTEX”, 10, point, RTNIL, NULL);
ads_invoke(in_rb, out_rb);

/* close feature */

in_rb= ads_buildlist ( RTSTR, “arc_featmake”,
     RTSTR, linetheme, -1, ename,
     0, “LINEEND”, RTNIL, NULL);
ads_invoke(in_rb, out_rb);

 /* To illustrate the creation of a normal feature,
consider the following example which creates a point
feature at 10,10 with a User-ID of 101. Please note
that this example also modifies the User-ID in the
```

```
attribute file. This example assumes that pointtheme
is a valid point theme with a valid attribute file.
*/

/* feature header */
point[0]=10.0;
point[1]=10.0;
point[2]=0.0;
in_rb= ads_buildlist ( RTSTR, “arc_featmake”,
     /* theme name */
     RTSTR, pointtheme,
     /* feature data */
     -1, ename, 0, “POINT”,
     10, point, 40, 0.0, 58, 101.0,
     /* item value group */
     -3, 1001, “esri”, 1040, 101.0,
     /* item name list */
     RTLB, RTSTR, “User_ID”, RTLE, NULL);
ads_invoke(in_rb, out_rb);
```

In the example above, it is not necessary to call the **featmake** function with a list containing ‘LINEEND’ because point features do not contain any subfeatures.

arc_featmod

Input	**Output**
featlist	`RTSTR` featname
itemlist	

Xloads required: arcad

This function modifies the data of a feature or subfeature. If the feature is successfully modified, the feature name will be returned as a `RTSTR`. If the feature cannot be modified, `RTNIL` will be returned. The name of the feature to be modified must be found in the group code 6 of *featlist*.

If *itemlist* evaluates to `RTNIL`, all item values in *featlist* are ignored. If *itemlist* does not evaluate to `RTNIL`, the item values for the feature will be modified. The values of items will not be modified for item names that are not in *itemlist*. In addition, the values are assumed to be in the same order as their item names.

Featlist is assumed to be in the same format as that returned by the **arc_featget** function. The only exception occurs when a polygon feature is to be modified.

The main features of polygons cannot be modified. Only the subfeatures of a polygon can be modified. The lists in the documentation for the **arc_featget** function show which group codes are modifiable and which group codes are read-only for line and point subfeatures of polygons.

It is important to note that the **arc_featmod** function cannot be used to change the number of vertices in a line feature. **arc_featmod** also cannot be used to modify the name of items or the item type in a record theme.

Example

```
/* This code fragment modifies the User-ID of the line
feature feat1
*/
struct resbuf   *in_rb, *out_rb, *new_rb;
double    dummy;
int       newid;

ads_getreal(“New User-ID: ”,dummy);
newid=dummy;

/* Get topology only */

in_rb= ads_buildlist (
   RTSTR, “arc_featget”, RTSTR, feat1, RTNIL, NULL);
ads_invoke(in_rb,out_rb);
in_rb=out_rb;
if (out_rb->restype==0 || out_rb->restype== -1) {
   while (in_rb != NULL) {
      if (in_rb->restype == 58) {
         in_rb->resval.rint= newid;
         in_rb= NULL;
      }
      else in_rb=in_rb->rbnext;
   }
   /* modify feature data */
   new_rb=ads_buildlist( RTSTR, “arc_featmod”,
   NULL);
   new_rb->rbnext=out_rb;
   ads_invoke(new_rb,out_rb);
   if (out_rb->restype == RTSTR)
ads_printf(“success\n”);
   else ads_printf(“failure\n”);
}
else ads_printf(“failure\n”);
```

Item access

Item manipulation functions are used to manipulate dBASE RECORD themes. They are also used to manipulate the attribute files of POLYGON, LINE, POINT and TIC themes. They perform the basic operations of creating, listing and retrieving item definitions. These functions cannot be used to modify the item values. The functions described in the ‘Feature selection set manipulation functions’ section (**arc_featget**, **arc_featmake**, and **arc_featmod** functions) can be used to modify the values associated with the items.

As explained in the introduction chapter, some feature classes have implied attribute files. The following list shows these implications.

Feature class	**Attribute file**
Annotation	None
Image	None
Line	AAT
Point	PAT
Polygon	PAT
Record (dBASE)	User defined
Record (SQL)	User defined
TIC	TIC

In other words, when you specify the *theme* name in the following item manipulation functions, the software automatically performs the operation on the implied attribute files. For example, if you use the **arc_itmdef** function (see next section for details) with a polygon theme, the function automatically defines the item in the theme’s implied PAT file (polygon attribute table).

Note: You cannot use item manipulation functions on annotation themes, image themes or SQL record themes. Annotation features are not spatially related to other features and therefore do not carry any implied attribute table. Annotation themes in ArcCAD are used only for annotating geographic features. Image themes do not contain spatial information, so there is no implied attribute table. An SQL record theme is accessed using ArcCAD software’s AutoLISP extensions and no ADS routines are available.

The item access functions described in the following section let you define database files (dBASE files) and add item definitions to store attribute information. For example, the **arc_itmdef** function lets you define item definitions. Similarly, the **arc_itmexi** function checks for the existence of the item in the data file prior to writing or retrieving item values. The **arc_itmget** function lets you access the item definition, and the **arc_itmlst** function lists all the item names in a theme's data file.

The following are some of the limitations that apply to item access functions:

- The maximum length of an item name is 10 characters.
- The only legal characters in item names are alphabetic characters, numbers and the underscore character.
- Item names cannot begin with a number.

Item manipulation functions

The following functions can be used to define, modify and list item definitions that are stored in database data files.

arc_itmdef

Input	**Output**
`RTSTR` theme	`RTSTR` itmname
-1 0	
0 "ITEM"	
1 itmname	
2 type	
70 column	
71 width	
72 ndec	
73 owidth	

Xloads required: arcad

This function adds a new item to a theme. If *itmname* is successfully created, *itmname* will be returned as a `RTSTR`. If *itmname* cannot be created, `RTNIL` will be returned. The function does not perform any user interaction. All of the parameters must be supplied. If user interaction is desired, use the **arc_cmd** function to invoke the user command that defines an item.

There are currently four item types available to the user: *Character*, *Date*, *Integer*, and *Numeric*.

Example

```
/* This code fragment defines the record theme
“LANDUSE” with items named ‘AREA’ and ‘LANDUSE_ID’
similar to that of a polygon theme
*/
struct resbuf   *in_rb, *out_rb;
int  i;
long ename[2];

ename[0]=0;
ename[1]=0;
i=1;
in_rb = ads_buildlist(
     RTSTR, “arc_thmdef”, -1, ename, 1, “landuse”,
     2, “record”, 3, “c:\\path\\landuse”,
     60, RTDXF0,
     NULL);
ads_invoke(in_rb,out_rb);
if (out_rb->restype==RTSTR) {
     /* define area */
     in_rb= ads_buildlist(
          RTSTR, “arc_itmdef”, RTSTR, “landuse”,
          -1, ename, RTDXF0, “ITEM”, 1, “area”,
          2, “n”, 70, 1, 71, 13, 72, 6, 73, 13,
          NULL);
     ads_invoke(in_rb,out_rb);
     if (out_rb->restype != RTSTR) i=0;
     /* define landuse_id */
     in_rb= ads_buildlist(
          RTSTR, “arc_itmdef”, RTSTR, “landuse”,
          -1, ename, RTDXF0, “ITEM”,
          1, “landuse_id”, 2, “i”, 70, 14,
          71, 11, 72, 0, 73, 11,
          NULL);
     ads_invoke(in_rb,out_rb);
     if (out_rb->restype != RTSTR) i=0;
     if (i==1) ads_printf(“success\n”);
     else ads_printf(“failure\n”);
}
else ads_printf(“failure\n”);
```

arc_itmexi

Input	**Output**
RTSTR theme	RTSTR status
RTSTR itmname	

Xloads required: arcad

This function checks for the existence of an item in a theme. If *itmname* exists in *theme*, a T will be returned as a RTSTR. If *itmname* does not exist in *theme*, RTNIL will be returned.

Example

```
/* This code fragment checks for the existence of a
user-defined item in a user-defined theme
*/

struct resbuf   *in_rb, *out_rb;
char theme[32], item[32];
ads_getstring(0,”Name of theme: ”,theme);
ads_getstring(0,”Item name: ”,item);
in_rb= ads_buildlist(
     RTSTR, “arc_itmexi”, RTSTR, theme,
     RTSTR, item,
     NULL);
ads_invoke(in_rb, out_rb);
if (out_rb->restype==RTSTR)
     ads_printf(“Item exists\n”);
else ads_printf(“Item does not exist\n”);
```

arc_itmget

Input	**Output**
RTSTR theme	-1 dummy
RTSTR itmname	0 “ITEM”
	1 itmname
	2 type
	70 column
	71 width
	72 ndec
	73 owidth

Xloads required: arcad

This function returns the definition of an item. If *itmname* exists in *theme*, a RESBUF containing the item’s name, type, width, number of decimal places, and output width will be returned. If *itmname* does not exist in *theme*, RTNIL will be returned.

Example

```
/* This code fragment retrieves the definition of the
item ‘AREA’ from the polygon theme ‘SAMPLE’
*/

struct resbuf   *in_rb, *out_rb;
int  start, width, ndec, owidth;
int  ok, i;
char type[4];

in_rb= ads_buildlist(
     RTSTR, “arc_itmget”, RTSTR, “area”,
     RTSTR, “sample”,
     NULL);
ads_invoke(in_rb,out_rb);
ok=1;
while (ok) {
     i=0;
     if (out_rb->restype== -1) i=1;
     if (out_rb->restype== 0) i=1;
     if (out_rb->restype== 1) i=1;
     if (out_rb->restype== 2) {
          i=1;
          strcpy(type,out_rb->resval.rstring);
     }
     if (out_rb->restype== 70) {
          i=1;
          start=out_rb->resval.rint;
     }
     if (out_rb->restype== 71) {
          i=1;
          width=out_rb->resval.rint;
     }
     if (out_rb->restype== 72) {
          i=1;
          ndec=out_rb->resval.rint;
     }
     if (out_rb->restype== 73) {
          i=1;
          owidth=out_rb->resval.rint;
     }
     if (i==1) {
          out_rb=out_rb->rbnext;
          if (out_rb==NULL) ok=0;
     }
     else ok=0;
}
```

arc_itmlst

Input	**Output**
`RTSTR` theme	`RTSTR` item name1
`RTSHORT` type	`RTSTR` item name2
	.....
	.....
	`RTSTR` item nameN

Xloads required: arcad

This function returns a list of the items defined in a theme's feature attribute table. If *theme's* feature attribute table contains items, a `RESBUF` containing a list of those item names is returned. If *theme's* table does not contain any items, or if the *theme* does not exist, `RTNIL` is returned. The *type* argument is optional, used to return only items of specific types.

Four item types are currently available: *Character*, *Date*, *Integer* and *Numeric*. Other item types available in dBASE, but not fully supported by ArcCAD, include *Logical* (logical data type) and *Memo* (MEMO field).

Note: ArcCAD functions do not currently support the creation of MEMO fields. However, if you use dBASE or other external programs to create MEMO fields, ArcCAD displays or lists them.

The optional *type* argument instructs **arc_itmlst** to return a list of items of a specific item type. The *type* argument is an integer (bit-coded) with values as shown below:

Bit value	Meaning
1	List *numeric* items
2	List *integer* items
4	List *character* items
8	List *date* items
16	List *logical* items
32	List *memo* fields

The bit values can be added in any combination to return a variety of item types. For example, if you wish to list *numeric* and *character* items, you can set the value of *type* to 5.

Caution: Future versions of ArcCAD might use additional **arc_itmlst** control bits, so avoid setting bits in your applications that aren't shown in the above table.

Notes

■ If the specified theme is currently related to another theme (established using the **relate** or **ddrelate** command), the related theme's items also displays. Items in the related theme are prefixed with a pound sign (#).

■ The **arc_itmlst** function cannot be performed on Annotation, Image, or SQL record themes.

Example

The following code fragments show some uses of the **arc_itmlst** function with the *type* argument. These examples assume that a theme named PARCELS exists in the current drawing and contains two user-defined items: OWNER (a *character* item storing the owner's name) and DATE (a *date* item storing the parcel registration date) in addition to the standard polygon attribute table (PAT) items AREA, PERIMETER, PARCELS_ and PARCELS_ID.

```
/* This code fragment returns a list of CHARACTER and
DATE items of a PARCELS theme.
*/

struct resbuf *in_rb, *out_rb, *cur_rb;
in_rb = ads_buildlist (
     RTSTR, “arc_itmlst”, RTSTR, “parcels”,
          RTSHORT, 12, NULL);
cur_rb = out_rb;
while (cur_rb->restype == RTSTR && cur_rb != NULL) {
     ads_printf (“Item name: %s\n”,
          cur_rb->resval.rstring);
cur_rb = cur_rb->rbnext;
}
ads_relrb (in_rb);
ads_relrb (out_rb);
```

Chapter 4

Command interface

cmd function

The **arc_cmd** function is a special function used to execute selected ArcCAD® commands from within ADS™. This command is similar to the **ads_command** function in standard ADS. Refer to the *AutoCAD Customization Guide* for further details on the **ads_command** function.

arc_cmd

Input	**Output**
RESBUF	RTNIL

Xloads required: arcad

This function executes ArcCAD commands from within ADS and always returns RTNIL. The **arc_cmd** function has a variable-length argument list. Arguments to the **arc_cmd** function are treated as pairs: the first of each pair identifies the type of the argument that follows, and the second, of the indicated type, contains the actual data. The first argument in the list is always the function name (**arc_cmd**) and is followed by a valid ArcCAD command.

The following example shows how to use the **defthm** command at the user interface level and an equivalent **arc_cmd** function to define a line theme.

```
     Command: defthm
     Theme name (?): roads
     Feature class: line
     GIS data set (?/<roads>): \demo\roads
     Line symbol number (0-100) (?/<0>): 3
     Command:
```

```
/* This code fragment uses arc_cmd function to create
a line theme named “ROADS” that uses a GIS data set
named “\demo\roads” and a line symbol 3.
*/
```

```
struct resbuf  *in_rb,  *out_rb;
in_rb  =  ads_buildlist  (
     RTSTR,    “arc_cmd”,
     RTSTR,    “defthm”,
     RTSTR,    “roads”,
     RTSTR,    “Line”,
     RTSTR,    “\\demo\\roads”,
     RTSHORT,       3,
     NULL);
ads_invoke (in_rb, out_rb);
```

An empty string (“ “) or `NULL` string is equivalent to entering a space on the keyboard.

arc_arcadver

Input	**Output**
`-none-`	`RTSTR` version number

Xloads required: arcad

This function returns a string that contains the current ArcCAD version number. Applications can tell what version of ArcCAD is being used by examining the string returned by **arc_arcadver**.

Example

```
/* The following code fragment returns the current
ArcCAD version number.
*/

struct resbuf *in_rb, *out_rb;

in_rb = ads_buildlist (
     RTSTR, “arc_arcadver”, NULL);
ads_invoke (in_rb, &out_rb);
if (out_rb->restype == RTSTR)
     ads_printf (“Version is %s\n”,
          out_rb->resval.rstring);
ads_relrb (in_rb);
ads_relrb (out_rb);
```

arc_dirlst

Input	**Output**
`RTSTR` path	`RTSTR` name1
`RTSHORT` type	`RTSTR` name2
`RTSTR` wildcard	`.....`
	`.....`
	`RTSTR` nameN

Xloads required: arcad

This function returns a combination of files, subdirectories and coverages in the current (or in a specified) directory. The *path*, *type* and *wildcard* arguments are optional. If the optional arguments are omitted, **arc_dirlst** returns a `RESBUF` containing a list of all files and subdirectories, including coverages, in the current working directory. Note that you can either ignore all the optional arguments or supply all the arguments. You cannot, however, omit only one of the arguments.

path: This optional argument is a valid pathname to the desired directory to search. A drive letter is permitted in the path, and you can use the forward slash instead of the backslash (but remember that you must use \\ to obtain one backslash in a string). A period (`'.'`) represents the current working directory.

type: This optional argument is an integer value (bit-coded) that filters the type of files to return. The following table explains the bit values and the data returned with each:

Bit value	**Meaning**
1	Return file names
2	Return subdirectory names
4	Return coverage names

The bit values can be added in any combination to return a variety of file types.

wildcard: The optional *wildcard* argument can be used to further filter the returned list. Only objects matching the wildcard pattern are returned. In the pattern, alphabetic characters and numerals are treated literally, a question mark (?) matches a single character, an asterisk (*) matches a sequence of characters, and certain other characters have special meanings within the pattern. Any valid AutoCAD wildcard string is accepted. Refer to the **ads_wcmatch** function in the *AutoCAD Customization Guide* for more details on wildcard options.

Example

```
/* This code fragment returns a list of files,
subdirectories and coverages in the current working
directory.
*/

struct resbuf *in_rb, *out_rb, *cur_rb;

in_rb = ads_buildlist (
     RTSTR, “arc_dirlst”, RTSTR, “.”, RTSHORT, 7,
          RTSTR, “*”, NULL);
ads_invoke (in_rb, &out_rb);
cur_rb = out_rb;
while (cur_rb->restype == RTSTR && cur_rb != NULL)
{
     ads_printf (“Name: %s\n”,
          cur_rb->resval.rstring)
cur_rb = cur_rb->rbnext;
}
ads_relrb (in_rb);
ads_relrb (out_rb);
```

Index