Advanced Customizations

Overview

The functions in this section can be used for many things, from making minor changes to a receipt format to making extensive changes to the system's functionality.

NOTICE:

These features are considered to be advanced development tools built in the program and are documented sufficiently for an experienced programmer or someone proficient with database design. Some sample forms and reports are provided which can be used as-is, modified for your needs, or studied as examples.

Our normal technical support will answer basic questions about the capabilities of these functions, but we cannot explain detailed examples or provide training for expressions programming, or debug changes that you've made. It can take some time to learn the details of these functions, even for experienced programmers, since they use a unique "language". If you don't have someone available with the time to experiment and learn these functions sufficiently to create the forms or reports you need, we can create forms or reports for an additional charge.

The "Advanced Customizations" functionality is considered an advanced customization system for Campground Master. Most users will be able to accomplish everything they need to without creating their own reports and forms. Before delving into these functions, we suggest that you thoroughly learn the rest of the system setup and investigate the other options -- there's a lot of flexibility already present in the various Tab View Options, Printing Options, Program Options, Data Field Definitions, Pick Lists, and Park Setup functions. If none of the normal options accomplish exactly what you want and you need to make a specific change to a tab view, receipt form, color coding or other functionality, then you can probably accomplish it through this section. However, we recommend that you do check the other options and/or contact us first to see if it can be done with normal options changes instead.

Samples

There are a number of sample Forms, Queries, and Color Schemes installed with Campground Master (in the Samples folder). These can be imported and used as-is, or you can make adjustments for your needs, or just look at them as examples. For details on importing the samples, refer to each of the following "Setup" sections according to what you want to import.

Tab Views Setup

The Tab Views in Campground Master are initially configured with all of the normal content views -- Rack, Arrivals, Departures, etc., and will include a Map and/or Query view if any maps or queries are set up. You can change the configuration of these views through Maintenance / Advanced Customizations / Tab Views. There are several reasons you might want to do this:

- To change the operator access levels for viewing or printing each view.
- To change the order, e.g. to make Map the first view so it's shown by default.
- To rename them to something more to your liking
- To make the names shorter to fit on a low-resolution screen
- To remove views that you never use
- To add multiple Query views showing specific queries by default

Note that changes to the tab view setup apply to all workstations.

Tab View Setup

The Tab View Setup dialog lists all of the tab views with their general information. Tabs are shown here whether they're enabled or not. If there's one that you don't plan to use, we recommend that you Edit it and uncheck "Enabled", rather than deleting it from the list. This way it's easy to re-enable later.

The order of views in the list will be the order they're displayed in Campground Master (skipping disabled ones of course). Note that any Map or Query views will only appear of one or more maps or queries are set up.

You can use the typical functions to add, edit, copy, move or delete tab views in the list.

Add the default tab views

This special function will add all of the default views to the list (which may result in duplicates). The main purpose of this is if you make some changes and then want to get back the original settings.

Note that if you delete or disable all tab views, the Rack view will still be shown by default (the program can't function without any tab views). If you re-enter Setup, it will prompt you to add all default views back in. It will also add all views back in automatically when you re-start the program.

Adding or Editing Tab Views

When you Add or Edit a view (or double-click on it in the Tab View Setup list), a simple dialog will allow you to enter the name, type, access levels and notes. You can also disable or re-enable it.

Tab Name

The name you enter here will be used as the header (name) of the tab, and also for the heading if you print the view.

Note that if you add tabs or change the names, you may not be able to see all of the tabs on the screen at once. If this happens, a pair of right/left arrows will be shown on the right side, which you can use to scroll the tabs back and forth. However it's usually better to get all of the tabs to fit on the screen. You can do this by shortening the Names of the tabs -- you'll notice that the tabs must all be the same size, so it uses the largest of the tab names to determine the size of all tabs.

Likewise, if you have extra space and prefer to widen the tabs, you can use longer names or even put spaces on each side of the longest name to force the tabs wider. You may already see this in the default "Payments Due" tab, if your screen is wide enough. Remember that these settings are used on all computers, so you'll have to keep the lowest-resolution screen in mind.

Content Type

The Content type is a selection list of the standard tab view types. The only special one is Query. In this view, you have the option to select a specific query to show. This essentially allows you to create a custom tab view using a query you've defined. If you select a query here, then only that query will be shown with no option to change it on the tab view. If you don't select a query, then the tab view will have a drop-down list where you can select a query. Thus you can have some "fixed" query tab views that show reports you need most, and also have a standard "Query" tab that allows you to dynamically select any query you've created.

Note that only one enabled entry is allowed for each of the types, except Query. There's no reason to add multiple Arrivals views, for instance, because they would be the same (they can't be set up with different options just because they're on different tabs).

Expressions

Overview

The "Expressions" functionality, or more specifically the expression processing engine, is the primary mechanism which allows extensive customization of forms, queries, reports, color schemes, and other elements of Campground Master. This is essentially a programming language with many built-in functions to access data, manipulate information, and generally get at any particular information you need. Since expressions are used in all of the advanced customization functions, you will need to get familiar with them in order to do your own customization.

In simple terms, an expression can be thought of as a calculation, or mathematical expression. If you're familiar with formulas in Microsoft Excel, or expressions used in Microsoft Access or dBase queries, then you'll have a big head start. For Scripting, familiarity with programming languages like Basic or C++ will help.

Expressions can be very simple or very long and complex, but every expression ultimately evaluates into a single value. This value can be a number, a string of text, a date, a boolean (true/false) value, or a data record pointer. The type of expression needed depends on the situation. For instance, boolean expressions are used in many places for filtering data or where conditional logic is needed like color schemes. Numeric expressions are used where calculated numbers are needed like transaction reports, and text expressions are used where text should be displayed such as receipt forms or query columns.

Macros are single expressions that can be called by name, with optional replacement parameters. They allow re-using common expressions, or simplifying things by calling a complex expression inside another expression. If you're familiar with Macros in the C programming language, they basically work the same way.

Scripts are a list of multiple expressions for performing more complex calculations, which can be called like a function from another expression. This is similar to defining a function in C or a procedure in Visual Basic or Pascal.

Expression Syntax

An **Expression** is simply a formula with values and functions, or at least one side of a formula, which evaluates down to a single value. The primary power of expressions comes from the built-in functions for extracting data values, converting and manipulating values, and making things happen.

An expression may contain the following elements:

Operators : +, -, >, etc.
Numeric values : 2, 5.3, .005, etc.
Text values : "Hi", 'a', etc.
Boolean values : .T. or .F.
Date values : {1/1/2005}
Function calls, often with parameters : Upper("hi")
Variables, user-defined : R, Total, etc.

The lowest-level format of an expression is a single value or function, or two expressions separated by an operator.

Here are some simple examples of expressions:

```
"Hello world"
2 + 2
Pi() * (R ^ 2)
Percent( 55/100 )
Resv:Resv_Adults > 2
iif( Resv:Resv_Adults > 2, "Extra adults present", "Two or less adults" )
FieldDate( ThisResv(), "Resv_First_Date" )
"Today is " + DateToText( Today() )
"Tomorrow is " + DateToText( Today() + 1 )
MessageBox( "Last name is " + FieldText( ThisCust(), "Cust_Last_Name") )
```

For a complete list of operators, functions, and other expression elements you can use, refer to the Expression Elements dialog. This is available anywhere expressions are entered so that you can build expressions by selecting from a list rather than memorizing functions or typing everything in.

Operator Precedence

As with any programming language, some operators have hight precedence, or priority, than others. This is the order of precedence, with the highest priority on top:

1. **+** or **-** unary operator (e.g. numeric prefix like +5, -10)
2. **NOT**, **!**
3. **^**
4. *****, **/**, **%**
5. **+**, **-**
6. **>=**, **<=**
7. **=, !=**, **<>**, **>**, **<**
8. **AND**
9. **OR**

Of course you can use parenthesis around expressions to force priority, like **((23 + 5) * (3-1)) % 3**.

Value Types

As hinted at above, there are several different types of values. Expressions are largely "type-safe", which means that in most cases you must use the expected type of value or else a parsing error will occur. For instance if a function requires a numeric value argument, then any other argument type will create an error. Also most operators must have matching types on each side (see the Syntax Rules).

Numeric values -- Integer or floating point numbers, using digits, a decimal an an optional minus sign in front. Do not include commas (e.g. 10,000 should be entered as 10000). Scientific notation (e.g 2.3E+9) is not allowed.

Character/Text values -- Anything enclosed in single or double quotes is considered text. There is no difference between single-character text an text strings. Quotes of the same type cannot be nested, but mixed single/double can be nested -- e.g. **"He said "Hi"!"** is not correct, but **'He said "Hi"!'** is correct. If you must use the same kind of quote in the text, it can be "escaped" with a backslash: **"He said \"Hi\"!"** will work. Also, text should not contain any control characters, like the return or linefeed characters. In some special places where long text is used such as in Forms definitions, the escape sequence **"\r"** can be used for return, and **"\n"** for line feed. Any other control characters can also be entered with hexadecimal escape sequences like **"\x07"**, but again keep in mind that thee are only of use in certain places like Forms definitions. Text values can be "added" together (concatenated) using the plus (+) operator.

Boolean values -- Also referred to as yes/no or true/false values. These are represented as the letter T or F with periods on each side (**.T.** or **.F.**). Keep in mind that this is also the way they would be displayed in a Query, for instance, if used as a raw boolean value. A simple way to convert these to Yes/No text is to use the **iif** function: **iif(value, "Yes", "No")** will return "Yes" if the value is true, "No" if it's false.

Date/time values -- There isn't a specific difference between date and time values -- technically one value contains both a date and a time. However the context in which it's used will determine whether the date or time portion is of interest. To enter a raw date in an expression, enclose it in curly-braces: **{4/15/06}**. A raw date value will also be shown this way, but you can convert a date to text using **DateToText** function. Dates are the one exception to the operator-type-matching rule. Adding a number to a date is an easy way to add (or subtract) a number of days. For instance, {4/15/06} + 10 = {4/25/06}.

Record values -- This is a special value type, in that records only have an internal representation (they're essentially an internal pointer), and therefore can be part of an expression as a result of a function or as a variable. The only valid operators for records are the equal and not-equal conditional operators. One common operation is to test for a record being "null", e.g. to see if the ThisSite context is valid. This must be done using the **NullRecord** function, like: ThisSite() != NullRecord(). Note that if a raw record value is displayed, only its record ID will be shown with a "#" prefix (you may see this when testing an expression in the Expression Creator dialog).

Unknown values -- This isn't really a "type", but you'll see reference to this in some function definitions. For instance, the **Macro** function has an unknown return value type because the result type will not be known until the macro expression is executed. Of course as the creator of the Macro you should know what the result will be, but the expression parser won't know. This can cause a parsing error if it's in an expression where a known type is needed, such as an argument to a function or on either side of an operator. In this case, you'll want to use one of the single-letter type-casting functions to force it to be a known type: **N( )** for numeric, **C( )** for character, **D( )** for date, etc. -- e.g. N(Macro("MyMacro")) if the macro is known to return a numeric value.

Expression Syntax Rules

Here are the basic rules for expressions:

- Function names are not case-sensitive (Today = TODAY)
- Function names must be immediately followed by the '(' character, with no space after the function name.
- There is no distinction between integer or floating-point numeric values. Internally they are all treated as floating-point, though some functions will only use the integer (truncated) portion.
- Parenthesis may be used around expressions to force the operator precedence.
- Anything not recognized as an operator, value or a function call is assumed to be a Variable, e.g. in **2 * R * Pi()**, "R" is assumed to be a variable.
- Spaces are not required around operators, values, or variables, but they are allowed (and recommended for readability).
- Text can be enclosed in single or double quotes (" or '). These can be nested inside each other to one level, if a function call requires a quoted expression (for instance, the following are OK: **Evalq("Evalq('Str(5)')")** or **Evalq("Str(5) + ' this is a 5.' ")**, but this is not: **Evalq("Evalq("Str(5)")")**
- If a quote is needed inside a text string, it can be escaped with a backslash: **"He said \"Hi\" to me"**.
- All expression elements have a result "type", e.g. numeric or text, and the result type is determined by the expression's final result. For instance, **2+2** is numeric, **2>2** is boolean, **"2+2"** is text. Some types cannot be mixed, e.g. **2 + "2"** is not valid, but numbers and dates can be added or subtracted: **{1/1/05} + 30** will add 30 days to the date 1/1/05.
- Any part of an expression not enclosed in quotes is parsed and evaluated, even if the result is not used. For instance in the expression **iif(2>3, 2+2, 2+4)**, all 3 expressions inside the iif() function are parsed an evaluated, even though the 2+2 expression is ultimately ignored. This is especially important when using functions which do things, like SetFieldValue(). You may not want it to "evaluate" all of them. This is where functions like iifq() come in handy, since the quoted expressions passed to it are just considered text except for the one that's used as the return value.
- There is no limit to the length of an expression or the levels of nesting, e.g. within function calls or parenthesis.
- There is a shorthand that can be used for some data field values, if used on the "Operational" record. For instance: **Resv:Resv_Type** is equivalent to **FieldText(ThisResv(), "Resv_Type")**. The shorthand versions are parsed slightly slower but are executed much faster. The Expression Elements dialog or Select Fields dialog will insert the shorthand version automatically when possible.

Expression Variables

You can pass information from one portion of an expression to another through variables. However a variable only exists as long as you're within the same Expression execution cycle where it's created, or at any level of CallScript, Macro, Eval, etc. executed within that same Expression. So it's unique to that Expression -- e.g. a variable created in a Query's Data expression will not exist for any other expression executed separately such as the Query's Filter expression (and thus will also not conflict with variables by the same name in other Expression spaces).

Variables are created with **SetVar( )** function -- they do not exist in the current expression's execution space before that. Within the SetVar function the variable name must be given as text, e.g. in quotes, like SetVar ("Index", 1). However when used in an expression, the name is not in quotes -- for instance: ThisListRec(Index). See the Scripts section for more examples.

Note that a variable used in an expression must exist when an expression is parsed, or else a parsing error occurs. Note the difference between these two expressions, which both eventually execute the two expressions contained as arguments:

```
Eval( SetVar ("Index", 1), ThisListRec(Index))
EvalQ( 'SetVar ("Index", 1)', 'ThisListRec(Index)')
```

The first expression will result in a parsing error because everything is parsed at once, and the variable "Index" used in ThisListRec() does not exist until execution time.

However the second expression will work because the arguments to EvalQ are only parsed as text initially. During execution, the first quoted expression containing SetVar is parsed and executed, creating the variable, and then the next quoted expression is parsed while the Index variable exists (it's in the same execution space).

Here are the rules for using variables:

- Variables can be of any type. Once the variable is created, its type is set also (to the type of the expression used in the SetVar function).
- Variable names must start with a letter and may contain letters or digits. They are not case-sensitive.
- Variables are short-lived, generally only for the current expression or expressions evaluated therein.
- Any variable that exists before a sub-expression is parsed and executed, such as in functions like CallScript, Macro, IIFQ, EvalQ, etc., is copied into the sub-expression and available therein, and its value is also copied back out.
- Any variable that does **not** exist before a sub-expression is parsed and executed but is created within the sub-expression will be destroyed before sub-expression returns (it's a local variable).
- These rules apply for any depth level of sub-expressions.

Expression Context

Expressions wouldn't be very useful if they operated in a vacuum, without any knowledge of what they're being used for. For instance on a receipt form, you couldn't use an expression to show the customer's name unless you know which customer you're printing the receipt for.

Expression "context" solves this problem by giving the expression processor as much information as possible about what the expression is being used for and what information is already known. This information is accessible through a number of "context" functions, or "This" functions.

The applicable information varies depending on the situation, but you can use the various "This" functions to get what's available. For instance in a list query's data expression, you can get the date range being requested for the query (**ThisFromDate** and **ThisToDate**), the base table of the query (**ThisTable**), and the specific record of the data line that the expression is being asked to fill (**ThisRecord**, or in some cases **ThisResv**, **ThisCust**, **ThisSite**, etc.).

Note that the **ThisRecord** function (or is short equivalent, **This**) is the generic form to get the current record in context. The record it returns could be any "type" of record, depending on the situation. This may be appropriate in general cases such as just checking for a valid record, e.g. **This() != NullRecord()**, or where there isn't a more specific function available. However in a case where you need a specific record type, such as a function to get a field from the record, you should use the more specific function like **ThisCust** or **ThisResv** whenever possible. This is especially true in Forms, where a single form may be useful in the context of Reservations, Customers, or even Sites. Using the specific context function allows you to easily create a copy of a Customer-based form and change the Base Table to "Reservations" so it can be used for reservations as well.

Refer to the applicable sections to find out which context variables are available in each situation.

Expression Processing

When an expression is used, there are two stages to its processing: Parsing and Execution. It isn't strictly necessary to know about this or understand it, but it can help in some cases to know the difference.

Parsing goes through the expression (which is actually just text to begin with) and converts the elements to an internal structure that's easy to execute. You might think of this as "compiling" in a language like C++ or Pascal. The syntax is evaluated for errors, function names are looked up, and it's converted to an operation tree.

Execution goes through the operation tree, calling any functions to evaluate the elements as needed, and evaluating the operators until it gets a single result for the expression.

In many cases, these simply happen one after another and it doesn't really matter whether it takes longer to parse or execute. However, in Queries where the program knows that the same expression is going to be needed many times (e.g. once for each record used in the report), it only parses the expression once. Then it does the execution of the expression many times, each time with a different record as the context record.

Sometimes you can use this to your advantage. For instance if you have a complex expression that involves some iif() conditions, you may not want it to execute both possible expression results in the function. In this case, it may work best to use the quoted-expression iifq() version. This will delay execution (and parsing) of the result parameters and only evaluate the one result that's needed.

Expression Errors

Whenever an expression is parsed or executed, error checking is performed. Depending on the function you're using, you may be able to see any errors encountered.

When using the Expression Creator dialog, you can "Calculate" the expression to check for basic syntax errors or other problems, to a limited extent.

When using the Save & Test function for Queries, Forms, Scripts, etc., any errors encountered may be shown as pop-up tips in a query or dialogs, or inside the form where the expression was to appear.

Expression Creator Dialog

The Expression Creator dialog is used in many places for entering expressions. It's invoked for "Add Expression" functions in queries, scripts, etc., and also available anywhere expressions can be edited.

In places where a dialog contains an expression editing field already, or shows an expression as static text, a button like "Test/Edit Expression" will open the Expression Creator dialog. Besides just a window for editing the expression, this has a few special functions to assist in creating the expression.

The Expression window acts like most other text entry windows, except that it also does some basic syntax highlighting. This colors the text according to the type of expression element. This happens dynamically as you type, as soon as the expression can be parsed into recognizable elements. The meaning of the colors is as follows:

Black -- Basic elements like math operators, variables and commas
Green -- Boolean operators (AND, OR, NOT)
Dark blue -- Numeric values
Violet -- Boolean values (.T. and .F.)
Dark Red -- Date values (eg. {1/5/2006})
Bright blue -- Recognized function names, once the parenthesis is present, like : Upper(
Magenta -- Text values
Bright Red -- Mis-matched parenthesis, braces or quotes (can't find the matching one)

You can type and edit the expression manually, or you can also use the Insert Expression Element button to select functions and operators to be pasted into the expression. This will open the Expression Elements dialog (explained below). If you have anything highlighted in the Expression window, the highlighted text will be replaced with the element selected (a prompt will warn you that this will happen).

One you have an expression entered, you can test it to a certain extent by clicking the Calculate button. (For that matter, you can use it as a fancy calculator this way). it will parse and execute the expression if possible. If there are no errors, the resulting value for the expression will be shown in the Result window. If there are errors, an Errors window will be added to the bottom and will list all errors encountered. If there are many errors, you can scroll to see the rest. (Often times one typing mistake will result in many errors as it continues to try to parse the whole expression.) Note that if the result is a record, the Record ID of the record will be shown (like "#000000002").

You can edit the expression and use Calculate as many times as you like. Once you're satisfied, click Save and the entire expression will be saved into whatever function you were editing the expression for.

Note that the Calculate function won't necessarily have all of the context information needed or variables defined to successfully execute the expression. If a base table is known, then it will use the first record in that table (e.g. the first reservation) as the operating record. Otherwise, most context information will not be valid for executing the expression here, which may result in errors even if the expression will work fine in its proper context.

Expression Timing

When you use Calculate, it will show the approximate time it takes to parse and execute the expression. If you're programming a complex report, this may help you make it more efficient by modifying the expression to execute more quickly.

Timing Tip: Most expressions take too little time to measure accurately just one time through. To get a better idea of the execution time, use the EvalQRepeatWhile function to make it repeat 100 or 1000 times. Note that the EvalQRepeatWhile function takes a quoted boolean argument and a count, so you need to modify the expression to be boolean (and always return True). One way to do this is to repeat the expression twice, once on each side of =, and use half the count. For instance, to measure the execution time of the expression Upper('abc') 1000 times, enter it like this:

```
EvalQRepeatWhile( "Upper('abc') = Upper('abc')", 500)
```

Note that with the EvalQRepeatWhile function, the expression in quotes is only Parsed once. Only the Execute portion of the processing is done multiple times. For places like queries where speed is important, execution is the time that's most important anyway.

Expression Elements Dialog

The Expression Elements dialog is used to look up and select elements to insert in expressions. All available functions, data fields, pick lists, operators and other expression elements can be found here. This dialog is primarily invoked from the Insert Expression Element button on the Expression Creator dialog, but is also accessible from similar buttons on dialogs that have an expression editing window built in, such as Edit Query Column and Edit Macro Definition.

When inserting an expression element, remember that any text selected in the expression editing window will be replaced with the element you select here. A prompt will warn you of this, so you can go back and un-select the text if necessary.

There are three lists in the dialog, which are used for looking up an expression element in a hierarchical fashion.

- First select the general kind of element in the first column, such as Function, Main Table Field, Pick List Value, Macro, or Operator.
- The other two columns will change depending on what you've chosen -- use the second column to narrow down your choice more, for instance by Function Type needed, or which Table you need the field for, or whether you need a Macro or Script, or which Pick List you need a value for.
- Finally, select the element in the third column, for instance the Field Name, Pick List Value, or Operator.

You'll notice when selecting an element that two windows will show information at the bottom. This information varies slightly depending on what kind of element is selected. Also note that what's actually inserted in the expression when you click Insert will depend on the kind of element.

Functions

All functions are shown by default (with "<All>" selected in the Function Type list), but you can narrow your search by selecting the Function Type of interest. Note that some functions will appear in more than one list, if it fits in more than one category.

The Return type and the Arguments needed for the function are shown in the first line, and a description of the function is shown below. Note that optional arguments are shown in brackets. These brackets may be nested -- for instance if it shows **nVal [,nVal2 [,nVal3]]**, that means the function could take one, two, or three values, and that nVal3 cannot be present without nVal2 also being present. As a general rule, arguments can never be skipped over -- to include an optional argument later in the list, all previous arguments must also be included.

When the Insert button is used to select a function for an expression, it inserts the function name and the required parenthesis, and also the argument list. This reminds you what arguments are needed, and then you can replace each argument as needed.

Note that the first letter of an argument indicates the type of argument needed (this is similar to what's called Hungarian notation, though it doesn't follow exactly the same rules as other programming languages).

n = Number
d = Date
c = Character, or Text, e.g. in quotes
e = Expression (may have various result types)
r = Record of the database, like a Customer record

You may notice that there are some duplicate functions -- two different function names that do the same thing. The main reason for this is to provide shortcuts for some functions are used frequently. For instance, **ThisResv()** can also be used as **Resv()**, and **ThisRecord()** can also be used as simply **This()**.

A description of each function can be seen when the function is selected in the Expression Elements dialog, so the details are not included in this printed documentation. However you can find the details of all of the functions listed in the online Help (press the F1 key when you're in the Expression Creator or Expression Elements dialog, and you'll find a link to the Function Reference near the bottom of the help screen).

Main Table Fields

This shows the five main tables that are most likely to be used, and all fields for the selected table are listed (only the fields that are enabled). The fields are in alphabetical order by the Field Name (according to Data Field Definitions).

The field's Description will be shown in the line below the selection lists, and the large window will show the shortcut field identifier that will be inserted into the expression. This is a 4-letter table abbreviation and the field ID, separated by colons.

Important: The shortcut field identifier is only valid if the context of the expression will have a "This" field of the selected table type -- e.g. if the base table is Reservations, then "Resv" is valid. If the base table is Customers, then "Resv" is probably not going to work.

All Table Fields

This shows all tables in the database (excluding pick list tables), and all fields for the selected table are listed (only the fields that are enabled). The fields are in alphabetical order by the Field Name (according to Data Field Definitions).

The field's Description will be shown in the line below the selection lists. The large window will show what will be inserted into the expression when Insert is used. This is generally a **FieldText** function to get the text of that field, using **FindRecByRecID** function to get a record of a known Record ID. This is just an example -- many times you can replace TableRecAt with a more appropriate function according to the context (e.g. **ThisListRec**). You may also want to use FieldValue instead of FieldText. The important thing is that it gives you the field name and the table name if needed, so you know how to get that field.

Pick List Fields

This shows all pick lists in the database, and the "fields" available for the pick list selected. The fields are in alphabetical order by the Field Name (according to Data Field Definitions). Otherwise it works like All Table Fields above.

Pick List Values

This also shows all pick lists, but instead of showing the fields it shows the Selection Names of all of the current pick list items. This is generally used if you're creating an expression to compare a pick list field value to one of the items in the list.

The numeric value will be shown in the first line at the bottom (which is actually the pick list item's record ID), and the pick list text will be shown in quotes below that. The text in quotes will be inserted into the expression, but you might want to use the numeric value in some cases (it may be faster to compare values than text, and isn't as likely to change as the text).

Fixed List Values

This works like the Pick List Values, but it shows all of the selection lists in the program that can't be changed like the Pick Lists. The most commonly used of these are the Reservation Status and Transaction Types.

Macros & Scripts

This allows you to select one of the built-in Macros, or a Macro or Script that you've defined yourself. When a Macro is selected, the windows below show the description (as defined in the Macro), and the complete Macro expression. When a Script is selected, only the description is shown.

When one of these is Inserted, it will also include the "Macro( )" or "CallScript( )" function call with the Macro or Script name filled in. In the case of a Macro, it will also include place holders for any arguments needed for the Macro.

Operators

This shows a list of valid operators of the selected type, and will show a description of the selected operator. Inserting an operator simply inserts the operator, with spaces around it. Spaces aren't usually required, but it makes the expression easier to read.

Macros

Macros are single expressions that can be called by name from within any other expression, with optional "replacement" parameters. They allow re-using common expressions, or simplifying things by calling a complex expression inside another expression. They basically work the same way as macros in the C programming language.

Macros Setup

To create a Macro, go to Maintenance / Advanced Customizations / Macros. This opens the Macro Setup dialog, which lists all current user-defined Macros and allows the typical functions for adding, editing, etc. Note that while there are functions to Move Up and Move down, their position in the list does not affect any other functionality -- it's mainly for your own preferences. However if you have many Macros defined, it may help the speed of your expressions to put the most speed-critical ones nearer the top, since they are searched in this order when executing expressions.

You can also Export one or more Macros to a text file, or Import Macros. This is primarily for you to import Macros created by the software provider, though it can also be used to transfer Macros between multiple databases.

Some sample Macros are included with Campground Master which are actually used by the sample forms. While these have specific uses for the sample forms, you can also look at them as examples. Click the Import macros button, and you'll get a typical Windows file dialog labelled "Import Macros". You need to locate the samples folder, which is typically in **C:\Program Files\Campground Master\Samples** (most likely you just need to double-click the "Samples" folder to get there). Now select the appropriate "Sample Macro" file, and click Open. Once the sample is imported, you'll see it appear in the list.

Note that the import/export files use the "CSV" file extension (e.g. Sample.csv), which means it's a comma-separated-value text file. Windows may recognize this file extension as something another program can open like Excel, but these are in a special format for importing records to Campground Master and should not be used in other functions. Also avoid opening different kinds of samples which use the same extension (e.g. don't open a Form sample from an Import Script function).

Be aware that Macros should not have duplicate names (or else it would just use the first one encountered by that name). If you make a Copy, text will be added to the name to make it unique. If imported Macros have duplicate names, the imported names are automatically changed to avoid duplication. This might affect the expressions where the Macros are used, so a warning will be shown and indicate which Macro name(s) were changed.

Note that Macro names are not case-sensitive. "MyMacro" is the same as "MYmacro".

Editing Macro Definitions

When editing a Macro, a simple Edit Macro dialog is used where you enter the Macro Name (which is used to invoke the Macro in an expression), the Description (which is shown if the Macro is selected in the Expression Elements dialog), and the Macro's definition itself. Buttons are available to Insert Expression Element and Test/Edit Expression, which invoke the corresponding dialogs to help build the Macro expression.

The Macro definition is just like any other Expression, with one difference. A Macro can have "Replacement arguments", and there is a special placeholder format you use in the definition to indicate where these arguments will go. The format for the placeholder is **#n#**, where the 'n' is the argument number starting at 1. So you may have **#1#** and **#2#**, indicating the first and second argument to the Macro. Here's an example:

```
iif( VarExists('#1#'), SetVar('#1#', #1#+1), SetVar('#1#',0))
```

This Macro takes 1 argument, which in this case is a variable name. The expression increments the variable by 1, or sets it to 0 if it did not exist.

Using Macros in Expressions

To invoke a Macro, you use the function **Macro( )**. Using the example above, assuming the Macro's name is "Inc", and the variable you want to increment is called MyVar:

```
Macro("Inc", "MyVar")
```

Note that the Macro name and all arguments must be text values. The argument's text that's inside the quotes, not including the quotes, will be inserted into the text of the Macro's expression, and then the expression will be executed. So in the above example, the Macro function results in this expression being parsed and executed:

```
iif( VarExists('MyVar'), SetVar('MyVar', MyVar+1), SetVar('MyVar',0))
```

There are a few important points to make about arguments for the Macro's expression:

- The Macro's expression, after replacing arguments, is actually parsed during the execution of the expression containing the Macro( ) function. This can greatly affect the speed of the parent expression.
- The replacement is strictly a text replacement, so the arguments can be anything -- an expression, a variable, or even an operator or function name. It can be any length, and could also be the result of another expression as long as it's a text result.
- Since the replacement may be done multiple times as in the example above, be careful what you use. For instance, don't use an argument that should not be executed more than once. Also keep in mind that it will need to be parsed many times (avoid huge expressions as arguments).
- Be careful about quotation marks in arguments, like " 'Text' ", if the argument is intended to be a text expression. While the example here is accurate for a text expression, the macro definition might also have quotation marks that conflict, creating an invalid expression (e.g. if the argument is enclosed in other quotation marks).

Macro Return Value

The "value" of the **Macro( )** function will simply be the value of the executed expression, which can be any value type. If this is used within a more complex expression, you might need to use a type-conversion function so the parser knows what type it's supposed to return. For instance, if you know the Macro will return text but the parser gives an error that it's an unknown type, use the "**C**" function to force it to text form. For example: **Upper(C(Macro("MyName")))**

Macros can be powerful tools when used correctly, but it's best to use them sparingly in places where speed is important. You could still create the Macros as a handy reference for commonly used expressions, but then copy the text out of the Macro definition into the expression where it's needed, instead of actually "calling" the Macro, so it doesn't delay the parsing until execution time.

Scripts

Scripts are mostly a list of multiple expressions for performing more complex calculations, with a couple extra flow control capabilities built in. Scripts can be called like a function from another expression, so they're similar to defining functions in C or procedures in Visual Basic or Pascal.

If you just need to do simple iterations like adding up transaction amounts, or just need to execute a couple separate expressions at the same time, then you can probably do it without a Script by using the "Flow" functions like **LoopSum( )** or **Eval( )**. For more complex tasks, though, a Script will allow more flexibility and will also result in a little "program" that's easier to read and modify than a large expression.

In many cases, a Script might also be more efficient because its flow control allows skipping the parsing & execution of expressions that aren't used. So even though you could probably do the same thing with carefully nested **IIF** and **Loop** functions, they might not be as fast.

Scripts Setup

To create a Script, go to Maintenance / Advanced Customizations / Scripts. This opens the Scripts Setup dialog, which lists all current Scripts and has the typical functions for adding, editing, etc. Note that while there are functions to Move Up and Move Down, a Script's position in the list does not affect any other functionality -- it's mainly for your own preferences. However if you have many Scripts defined, it may help the speed of your expressions a tiny bit to put the most speed-critical ones nearer the top, since they are searched in this order.

You can also Export one or more Scripts to a text file, or Import Scripts. This is primarily for you to import Scripts created by the software provider, though it can also be used to transfer Scripts between multiple databases.

Some sample Scripts are included with Campground Master which are used by the sample forms. While these have specific uses for the sample forms, you can also look at them as examples. Click the Import scripts button, and you'll get a typical Windows file dialog labelled "Import Scripts". You need to locate the samples folder, which is typically in **C:\Program Files\Campground Master\Samples** (most likely you just need to double-click the "Samples" folder to get there). Now select the appropriate "Sample Script" file, and click Open. Once the sample is imported, you'll see it appear in the list.

Note that the import/export files use the "CSV" file extension (e.g. Sample.csv), which means it's a comma-separated-value text file. Windows may recognize this file extension as something another program can open like Excel, but these are in a special format for importing records to Campground Master and should not be used in other functions. Also avoid opening different kinds of samples which use the same extension (e.g. don't open a Form sample from an Import Script function).

Be aware that Scripts should not have duplicate names (or else the **CallScript** function would just use the first one encountered by that name). If you make a Copy, text like "(copy 1)" will be added to the name to make it unique. If imported Scripts have duplicate names, the imported names are automatically changed to avoid duplication. This might affect the expressions where the Scripts are used, so a warning will be shown and indicate which Scripts name(s) were changed.

Script names are not case-sensitive, so "My Script" will appear to be the same as "my script".

Editing Script Definitions

When editing a Script, an Edit Script dialog is shown with each line of the script (each line being a single expression), and also a place to enter the Script Name (which is used to invoke the Script in an expression), and the Description (which is shown if the Script is selected in the Expression Elements dialog).

Creating a Script is simply a matter of adding one line (expression) after another, just like writing a program. Of course you can Insert lines, move and copy lines, etc. as needed. Each time you Add, Insert, or Edit an expression line, the Expression Creator dialog is used. Thus you can test-calculate each line, look up functions, etc. like anywhere else.

Adding Multiple Lines

One special function available for editing Scripts is Add Multiple Lines. This opens a plain multi-line text window where you can enter as many lines as you want, all at once. This is usually faster than entering them separately if you don't need to look up functions, etc. through the Expression Creator. More importantly, it lets you copy/paste the lines from somewhere else, e.g. if you prefer using a different editor for programming. Just make sure that each expression has a "line feed" or "return" after it (long expressions will wrap but should not have the line feeds in them). Once you've entered or pasted what you want, click Save. Each line will be added as a separate line in the Script.

Save & Test Script Results

This function simply uses the Expression Creator dialog, with a CallScript function already inserted to call the script you're working on. Press Calculate to execute the script to check for errors. Of course, like any other expression, it won't necessarily have all of the context information needed or variables defined to successfully execute the script, but you may be able to catch some of the basic problems with the script format, like mis-matched IF/ELSE/ENDIF sets or WHILE/ENDWHILE pairs.

Using Scripts in Expressions

To invoke a Script, you use the function CallScript( ). Note that the only argument is the script name itself (in quotes of course, since it's a text argument).

```
CallScript("MyScript")
```

Note: Script names are not case-sensitive. "MyScript" is the same as "MYscript".

There are no optional arguments available to be passed into the Script. If you need information passed into the Script, you can use variables (see below).

Scripts can be called within other Scripts -- even the same Script can be called recursively.

Script Return Value

Each Script has a single return value, just like any other expression (technically this becomes the return value of the CallScript function). To return a value from a script, simply make sure that value is the result of the last line of the script. Remember that script lines are just expressions executed one after another. So the "value" of the last line of the script will be the returned value -- often times this is simply a variable name, if the Script sets the variable to the desired value.

The return type can be any type of value. Note that if the Script is used within a more complex expression,

you might need to use a type-conversion function so the parser knows what type it's expected to return. For instance, if you know the Script will return a number but the parser gives an error that it's an unknown or mis-matched type, use the "N" function to force it to text form. For example: **N(Script("JanReceipts")) + N(Script("FebReceipts"))**.

Script Variables

It was mentioned before that you can pass information into a Script using variables. You can also pass information out of a Script through variables, for instance if you need more than just the single return value. Here are the rules for using variables for Scripts:

- Variables are created with **SetVar( )** function -- they do not exist in the current expression's execution space before that.
- Variables exist as long as you're within the same Expression execution cycle where they're created, or at any level of CallScript, Macro, Eval, etc. executed within that same Expression. They are unique to that Expression -- e.g. a variable created in a Query's Data expression will not exist for any other expression executed separately such as the Query's Filter expression (and thus will also not conflict with variables by the same name in other Expression spaces).
- Any variable that exists before the Script is invoked with **CallScript** is copied into the Script, and its value is also copied back out (it's a global variable, which can be changed inside the Script).
- Any variable that does not exist before the **CallScript** but is created within the Script is destroyed before **CallScript** returns (it's a local variable). It will exist for the life of the Script, since each Expression line is parsed and executed within the same execution space.
- These rules apply for any depth level of script calling -- e.g. every CallScript creates a new level of variable scope, with any existing variables copied into it and back out of it.

Script Flow Control

There are two directives available for flow control -- **IF** and **WHILE**. Script lines starting with these are specially handled (pre-parsed), not just executed as an expression. Any text following the IF or WHILE directive (separated by a space) is parsed and executed as an expression, and must have a boolean result value. The result of course determines whether the lines following the IF or WHILE are executed.

Both of these directives must be followed in the Script by a line containing **ENDIF** or **ENDWHILE**, respectively. In the case of IF, you can also have an **ELSE** directive before the ENDIF. There is not an ENDELSE -- the ENDIF is used to terminate both the original IF prior to the ELSE directive.

These may be nested to any level, but each directive set must be properly nested. Any directive set that's not properly nested will result in an error.

These directives are not case-sensitive (e.g. "if" or "If" may be used), but we suggest using upper-case for ease of script readability. See the examples below.

Indentation & Spacing

Leading spaces in front of any text in line are ignored, so you can put spaces in front of any expression (including comments and flow control directives). We recommend indenting each level of IF or WHILE nesting by at least 3 spaces to improve readability. You can also insert blank lines to separate sections. The excerpts below show a couple examples of nesting and the use of flow control directives, and comments in scripts.

```
SetVar( "nResv", NumTran( ThisResv() ) )
SetVar( "bFound", .F. )
SetVar( "r", 1)
; Verify that this reservation contains this transaction
WHILE r <= nResv AND !bFound
   IF ResvTran(r) = ThisTran()
     SetVar( "bFound", .T.)
   ENDIF
   SetVar( "r", r+1)
ENDWHILE

.......

IF CCNum != ""
   REM always need at least 2 lines
   SetVar("nLines", 2)

   IF C(Macro("TranCCAuth", "ThisTran()")) = "Y"
      SetVar("nLines", nLines+1)
   ENDIF

   // Need more lines if a signature is shown
   IF SettingLocalBool("Print", "PrintShowCCSigLine")
      SetVar("nLines", nLines+7)
   ELSE
      SetVar("nLines", nLines+2)
   ENDIF
ENDIF
```

Script Comments

It's a good idea to include comments in a complex Script, so you can understand it later. Any "expression" line starting with a double-backslash ("***//***") or a semicolon ("**;**") is considered a comment. You can also use "**REM** " like in Basic (there must be a space after "rem", but it doesn't have to be upper-case). Don't worry that you're using the same Expression Creator to ener comments -- just enter the text the way you want it as if it's an expression. A comment cannot be on the same line as an expression.

Function Reference

Function Types

In the Expression Elements Dialog where you can browse for functions and select the from a list, the functions are divided into several categories. This is simply to help narrow down your search -- there isn't necessarily a difference in the way each category is used. You'll also notice that some functions appear in more than one list, e.g. if their function involves more than one category.

Each category, or type, is briefly described below. The details of each function are not included in this printed documentation, but you can find the details in the online Help (press the F1 key when you're in the Expression Creator or Expression Elements dialog, and you'll find a link to the Function Reference near the bottom of the help screen).

Context (This...)

These functions are special in that the access values related to where the function is used. See the previous Expression Context section for more details.

Conversion

These functions are used to convert one type of value to another (e.g. numeric to text, text to date, etc.)

Database

These functions are used access any kind of database information (but not change it). In addition to accessing raw data fields, etc., many functions are included to get commonly needed information, for instance the current operator or the total amount due for a reservation.

Database Modification

These functions are used to modify the database. Naturally these should be used with care, since modifications cannot be simply undone, and anything modified through functions is likely to be done without the user knowing it has happened.

Date & Time

These functions are related to getting the date or time, or manipulating date or time values.

Financial

These functions are related to financial operations, such as properly rounding or formatting amounts for your currency.

Flow

These functions allow you to conditionally execute other functions, thus they can be used to affect the "flow" of the expression.

General & System

These functions give you access to general information about Campground Master, or things outside of Campground Master such as Windows functions and files.

Inspection

These functions are used to "inspect" or "test" the results of an expression, generally for use in conjunction with Flow type functions.

Math

This function category includes anything related to basic mathematical operations.

Settings

These functions access or change program settings. While the settings are actually contained within the database, these functions are separated for convenience in looking them up.

Text

This category includes any functions related to manipulating text value types, for instance to extract parts of a text string.

User Interaction

Functions that involve some kind of interaction with the user are included in this category, such as showing message prompts or opening a particular user-interface dialog like Reservation Details.

User-defined Dialogs

All of these functions are specifically for use in Dialogs definitions, to access or manipulate the controls on a dialog (and to open a user-defined dialog).

Color Schemes

A Color Scheme is a set of rules used to determine the color of an item in a grid or an indicator on the map view. It can define the specific conditions for any number of foreground / background color combinations.

There are several places where color schemes are used. Primarily they're used for the color of data in Query columns, but they can also be used for special coloration of the heading rows or columns of Queries or on the standard tab views like the Rack. A color scheme can be used to add custom color combinations to the standard colors used on the Rack (and other tab views where a reservation name is shown), or on the Map.

Color Scheme Setup

To create a Color Scheme, go to Maintenance / Advanced Customizations / Color Schemes. This opens the Color Schemes Setup dialog, which lists all current schemes and has the typical functions for adding, editing, etc. Note that while there are functions to Move Up and Move Down, a scheme's position in the list does not affect any other functionality except for the order they will appear in selection lists -- it's mainly for your own preferences.

You can also Export one or more schemes to a text file, or Import schemes. This is primarily for you to import Color Schemes created by the software provider, though it can also be used to transfer schemes between multiple databases.

Some sample Color Schemes are included with Campground Master, which you can use or learn from (some may also be used in sample Queries). Click the Import schemes button, and you'll get a typical Windows file dialog labelled "Import Color Schemes". You need to locate the samples folder, which is typically in C:\Program Files\Campground Master\Samples (most likely you just need to double-click the "Samples" folder to get there). Now select the appropriate "Sample Color Scheme" file, and click Open. Once the sample is imported, you'll see it appear in the list.

Note that the import/export files use the "CSV" file extension (e.g. Sample.csv), which means it's a comma-separated-value text file. Windows may recognize this file extension as something another program can open like Excel, but these are in a special format for importing records to Campground Master and should not be used in other functions. Also avoid opening different kinds of samples which use the same extension (e.g. don't open a Form sample from an Import Script function).

Color Scheme Names

Note that Color Schemes cannot have duplicate names. If you make a Copy, text will be added to the name to make it unique. Color Scheme names are not case-sensitive.

Also note that even though Color Schemes are usually selected by name, for instance when defining a Query, the record link to the scheme is used to reference it internally. Thus you can change the names of schemes already in use without affecting any function that uses the scheme. However this also means that a scheme that's already in use (linked from a Query) cannot be deleted.

Editing Color Schemes

To edit a Color Scheme, use the Add or Edit functions in the Color Scheme Setup dialog. Other functions where schemes are referenced, such as the Global Color Schemes function, have buttons to directly edit the scheme without leaving that function and going through Color Scheme Setup.

When editing a scheme, an Edit Color Scheme dialog is shown with each Color Rule of the scheme (each rule being a single expression and the colors represented), along with a place to enter the Color Scheme

Name, and an optional Default Scheme.

Creating a Color Scheme is simply a matter of adding Color Rules. Of course you can Insert rules, move and copy rules, etc. as needed. When a rule is Added or Edited, the Edit Color Scheme Rule dialog will be used to enter the details.

Editing Color Rules

The Edit Color Scheme Rule dialog is invoked from Edit Color Scheme, when adding or editing rules.

Here you enter a Name for the rule, select the foreground and background colors to be used, and enter an Conditional Expression which defines when the rule is to be used. You can type the expression directly in the dialog here, or use Insert Expression Element to select and insert elements as needed. You can also use Test/Edit Expression to open the Expression Creator dialog. The Expression Creator has basically the same functions as the Color Rule dialog, but it also has the Calculate function for testing the expression.

Note that the Name should be as descriptive as possible, explaining what that color rule specifies. If this Color Scheme is used as a Global Default Scheme for reservations or maps, then the name of each rule will be shown in the Color Key list.

You'll notice that each rule has both a foreground and a background color. You can't just define a rule for the foreground, e.g. to make any unpaid reservation have red text. You must define rules for every possible combination of colors (at least any that won't be covered by the scheme's "Default scheme").

How Color Schemes Work

When a color scheme is used for a data element, the program executes the Condition Expression for each Color Rule in sequence. If the expression results in a True (.T.) value, then the color combination selected for that rule is used. No further rules are processed for that data element. If the expression results in a False (.F.) value, then it continues checking the following color rules until a True value is found.

If none of the rules result in a True value, then it checks for a Default scheme. If a Default scheme is defined for the Color Scheme, then that will be used to determine the data color. If not, then the program will probably use whatever default is normally used for the data (e.g. as if no color scheme was specified). If you want to specify your own default color so that it never uses the program's default, then add a rule to the end with just "**.T.**" (without the quotes) as the Condition Expression. That will force the last color to be used (since the condition is always True), if none of the previous colors apply.

Functions for Color Scheme Expressions

The context is important in most color schemes, since that's usually the basis for the color rules. For instance, if the scheme is to be used for coloring Reservations, then the **ThisResv()** context function will be heavily used in the scheme's rules. The context functions **ThisCust(), ThisSite()** and **ThisPark()** would also be available. If it's a scheme to be used in a Transactions query, then **ThisTran()** is also available.

The date can also be important for color schemes, e.g. to show on the Rack whether it's paid up to the given date (and scheduled Period if applicable). The context functions **ThisDate()** and **ThisPeriod()** are used to get the specific Rack column date other date of interest, for instance the "as-of" date on Payments Due.

In some cases, such as the data color schemes for Cross-table Queries, other context functions are available to get the appropriate information such as **ThisValue()** or **ThisGroupText()**. See that section for more details.

There are also some special functions that can be useful in color schemes. Remember that each rule needs to check for a precise condition for the color combination of interest, and yet those conditions should be unique enough not to use the color for the wrong things. For instance if you want to define a special color for Hourly reservations, you need to do much more than just check the Reservation Type field. You would probably want several color rules like there are for other types, to indicate different status for hourlies like pending, confirmed, guaranteed, checked in, etc. You should also make sure these colors don't accidentally apply to special cases like conflicting, cancelled, checked out, etc.

So you can see that the expression for a rule might need to check quite a few things, which not only takes your time to create the expressions for but also slows down the processing of the rules. Therefore some shortcut functions have been added to get a value for the "default" color status that would otherwise be used for Reservations (**ColorStatusResv**), Map sites (**ColorStatusMapSite**) or Rack sites (**ColorStatusSite**). These return a numeric value based on one of the color-status values in the default color keys, which you can use in a Color Rule expression to determine what color would normally be used.

To get the values returned for each function , go into the Insert Expression Elements dialog, select "Fixed List Values", and select either "Site Status (color default)" or "Map Status (color default)". When a status is selected in the last column, its numeric value will be shown in the box below.

So now you can create a rule that overrides a normal color with an expression that compares the normal status and your specific condition, rather that an expression that has to check several different things. For example to override the "Resv., Pending, Don't Move, Paid" color for normal reservations to be a different color for hourlies would normally require checking at least 5 different fields for the precise condition, but you can do it with this expression:

```
ColorStatusSite( Resv(), ThisDate(), ThisPeriod() ) = 37
AND ResvBaseType( Resv() ) = "Hourly"
```

Note that the expression above is shown on two lines for clarity -- the expression itself should not be broken into two lines, but of course it may wrap automatically in the dialog's edit box.

For site-based color schemes for instance on the Rack (particularly if used for a Default Global Color Scheme), use **ColorStatusSite**. This will return one of the "Site Status" colors for the reservation(s) on the site if applicable, or the open site, for the given date and period. For a color scheme that's to be used only on a Reservation-based Query, you can use **ColorStatusResv** to get the status of a reservation as of a particular date (which may affect paid/not-paid, etc.).

If you create a color scheme to be use as the default on the Map view, use **ColorStatusMapSite** so that it handles the special map coloring for the site (and also provide the date arguments for the map range, which will be available in the context functions **ThisFromDate** and **ThisToDate**). If you're trying to color the open sites according to an attribute, for instance, be sure to have the expression check that the site would normally be shown "open" for the map as well as for the attribute. For instance to have a color rule that highlights 50 Amp sites in a different color, use the expression:

```
ColorStatusMapSite( Site(), FromDate(), ToDate()) = 1 AND Campsites:Attrib_50A
```

Note that a Color Scheme to be use for the Map's Default Global Color Scheme does not need to have a "Default scheme" -- it's assumed to use the Map's normal colors by default.

Default Global Color Schemes

This function is accessed through Maintenance / Advanced Customizations / Select Global Color Schemes. Here you can select a custom color scheme to be used by default for Reservation coloring, Map indicator coloring, open sites on the Rack or the various Rack heading rows and columns.

While this isn't the normal place to set up Color Schemes, you can use the Edit Scheme buttons to edit any selected scheme or to create a new one (if no scheme is currently selected).

Note that the color scheme selected here for the Reservations override will be used on the Rack (for sites occupied by a reservation) as well as on the other tab views where the customer's name is shown. It will also be used on the Map view if the "Use reservation color coding" option is selected there. In other words, it's used anywhere that a reservation is typically shown in a color to indicate its status. Any color scheme used for this should have "Reservations" selected as the Default scheme. In addition, the Color Key function will include the colors from the scheme, and also allows editing the colors (or the entire scheme) directly from the Color Key dialog.

The Rack headings color scheme selections can be used to override the normal static heading colors on the Rack. This can be used for instance to show the Site names or Types in different colors depending on their status, Park, or any attribute. Any scheme for site headings or open sites will use the context function **ThisSite** available, which can be used in the expressions to get the appropriate site information.

You could use a color scheme on the date headings to highlight certain days of the week, holidays, or any special situation. A special color scheme can also be used for any open sites on the Rack, for similar reasons or to indicate something about the site. There is a special context function, **This Date**, which indicates the date of the cell or heading of interest in these schemes. Here's a simple example of an expression which can be used in a Color Rule to change the color of the header for Saturday and Sunday:

```
DoW(ThisDate()) = 7 OR DoW(ThisDate()) = 1
```

Note that the default color scheme selections are also global in that they affect all workstations.

Queries

Overview

Queries are at the core of most reporting functionality. At the most basic level, a Query is defined as a data filter and a number of data extraction (field selection and formatting) specifications. In Campground Master, a query is typically represented by a grid with data in it.

Any of the Tab views (except the Map view of course), and just about every report or grid view available in Campground Master could be considered a Query even though they're built-in functions. Although these are programmed internally for the benefit of speed, you could reproduce any of them using the Query definition functionality. In some cases you may want to do just that, with minor modifications to suit your own specific needs. However if it's one of the Tab views you want to modify, there's no need to recreate the whole Query -- just create a Query with the special information columns you need to add on to the standard columns available, and select that Query as an Add-on Query in the Tab's Option function.

When any Queries are defined, you may also have a Query Tab view. Here you can select any query to be shown on the Tab, and optionally select start and end dates for filtering and possibly enter search text to use in the filtering (making it a kind of "Find" function).

Query Types

There are two general types of Query in Campground Master -- List Queries and Cross-Table Queries.

A **List Query** is like the Arrivals Tab View -- it's a list of records (specified by a filter definition) and a number of Query Column definitions which specify what fields or calculated information to show for each record. The column definitions can include custom formatting and color coding by Color Schemes (including the headings), and custom sorting rules. You can also select an expression to be shown for a pop-up tip (when the mouse is held over a cell), as well as a double-click action. This makes them interactive, and thus more useful than just a static report.

A **Cross-Table Query** is like a Transaction summary report or the Rack view -- it's also generated from a list of records, but instead of showing a record in each column it shows calculated information in a cross-table, where the records used for the calculation of each cell of the table are filtered by two additional conditions -- one for rows and one for columns. You can specify nearly any kind of grouping you want for both the rows and the columns, making it easy to cross-correlate information in various combinations. This allows for much more creativity than just showing information by date.

Queries Setup

To create a Query, go to Maintenance / Advanced Customizations / Queries. This opens the Queries Setup dialog, which lists all current Queries and has the typical functions for adding, editing, etc. Note that while there are functions to Move Up and Move Down, a Query's position in the list does not affect any functionality other than its order in drop-down selection lists. This of course may be important to you for organizational purposes.

You can also Export one or more Queries to a text file, or Import Queries. This is primarily for you to import Forms created by the software provider, though it can also be used to transfer Forms between multiple databases. Note that the Export and Import functions will also Export or Import any Color Schemes used in the Query definition.

Queries cannot have duplicate names (or else they could not be uniquely selected from a list). If you make a Copy, text like "(copy 1)" will be added to the name to make it unique. Of course you can change this to something more appropriate. Duplicate checking for the names is not case-sensitive ("My Query" is considered the same as "my query").

When you Add or Insert a query, you will first be asked what kind of Query you want. This is because the editing of each type is quite different. Once you select the type, it will continue to an editing dialog for that type -- either for a List Query or a Cross-Table Query.

Importing Sample Queries

Several sample Queries are included with Campground Master which you can use as reports, learn from, or modify as needed. To use these samples, you must first Import them. Click the Import queries button, and you'll get a typical Windows file dialog labelled "Import Queries". You need to locate the samples folder, which is typically **C:\Program Files\Campground Master\Samples** (most likely you just need to double-click the "Samples" folder to get there). Now select the appropriate file, for instance "Sample Query - Rent Roll" to get a Rent Roll example, and Open.

Note that the import/export files use the "CSV" file extension (e.g. Sample.csv), which means it's a comma-separated-value text file. Windows may recognize this file extension as something another program can open like Excel, but these are in a special format for importing records to Campground Master and should not be used in other functions. Also avoid opening different kinds of samples which use the same extension (e.g. don't open a Form sample from an Import Script function).

When you're importing sample Queries, it may also import Macros or Scripts that are used in the Query. If these are already defined, resulting in a duplicate name, then a warning will be shown listing the duplicates and what their names were changed to during the import. These might be safe to delete, assuming the imported version does the same thing as the original version. Otherwise you will need to change any expressions in the Queries that use the Macro or Script so that it uses the correct name.

Once the sample is imported, you'll see it appear in the Queries list. You can Edit the Query to make any changes you need.

Editing List Queries

The Edit List Query Definition dialog is shown when adding or editing list queries from Queries Setup. Other functions where Queries are referenced, such as the Queries Tab view, may also have a button to directly Edit the Query without leaving that function and going through Queries Setup.

Here you can edit all of the components of a List Query. There are a few fields you can edit directly, a couple of buttons for editing the list of Default Sorting and Filter Conditions for the Query, and the main portion of the dialog for editing the "meat" of the Query -- the Query Columns. The order of the columns is also defined here, by their order in the list, which can be adjusted easily with the Move Up and Move Down buttons.

Query Name

The name should be descriptive enough for selecting the Query out of a drop-down selection list. Queries will usually be shown in the order they appear in Queries Setup, not alphabetical, so the name doesn't affect the order. Each Query must have a unique name (which is not case-sensitive).

Base Table

The base table determines the primary data table of the Query -- that is, which records are potentially going to be included in the Query. For instance, if the base table is Reservations, then the Query will include all Reservation records by default, subject to the Filtering Conditions.

Selecting the base table is important, but it's not always an obvious choice. For instance, lets say you want a list of outstanding balances. If you choose "Customers" as the base table, then you could easily show customer balances but it wouldn't be as easy to show reservation information (site or in/out dates, for instance), and the balance might not represent a current reservation. If you choose "Reservations" as the base table then you can show each reservation with a balance, and any related information, and it would show multiple lines for the same customer if they have more than one site reserved (e.g. linked reservations).

Also keep in mind that the base table affects which context functions are available for Query expressions. If "Customers" is the base table, then only ThisCust is useful (because a Customer record doesn't necessarily have a unique reservation). If "Reservations" is the base table, then you can use ThisResv, ThisCust, ThisSite, and ThisPark, because all of that information is known for each reservation.

Another function of the base table selection is to determine where a Query is appropriate. For instance, Query must have a base table of Reservations to be added to tab views like Arrivals, Departures, etc., and a base table of Transactions to be added to the Transactions tab view. It must have a base table of Sites to be added to the Rack sites headings.

Any table in the database may be selected, but only a few are useful for most situations -- Reservations, Customers, Transactions, and Sites.

Note that if you're creating a Query to use with the date headers of the Rack, then the base table does not matter since no record information is used. However for testing reasons, we suggest using Parks just because the Save & Test function will try to show all records of the selected base table, and the Parks table usually only has a few records.

Access Level

The access level simply determines which operator access level is required to view the Query. If the current operator does not have the selected access level, then the Query will not be shown in the selection list on the Queries Tab view, for instance.

Exclude from Lists

When this is checked, the Query will not be shown in any selection lists such as in the Queries Tab view. Note that this does not actually disable the Query -- e.g. if it was already selected as an add-on for a tab view then it will still be used in that function. However it won't be available as a general selection, which can be handy for keeping special-purpose Queries out of view. Of course this also means that you would need to uncheck this again (temporarily at least) if you ever did need to select the Query somewhere. Another alternative would be to set the Access Level for such Queries to Administrator, assuming that most operators are non-administrator.

Note that one place that Exclude from Lists does not affect is the Tab Views Setup (since this is assumed to be an administrator-only function anyway).

Default Sorting Hierarchy

The default sorting hierarchy is used as the lowest-order sorting when any column is sorted in the query, if the specific column sorting results in equal values.

For instance, if a Transactions-based query is sorted by Site, you may have a lot of records with the same Site. The normal Site column sorting would consider these equal, so the order of those records of the same site could be random. However if you've specified a default sorting hierarchy, then it would be used to further sort those records of the same site -- in the case of Transactions, we suggest using a default hierarchy that includes the Date, Time, and Record ID fields of the transactions, in that order.

Filtering Conditions

The filtering conditions determine which records of the Base Table are included in the Query. Without any filtering conditions, ALL records are included (which in some cases is what you want, e.g. to include all Sites). Refer to the Filter Conditions section below for more details.

Important: If this query is selected as add-on for another Tab View (e.g. Arrivals), then any filtering conditions defined here are also used to determine which records are shown. Only records that pass the filtering here ***and*** the tab view's normal filtering will be shown.

Includes Text Filter

When this option is checked, the user will have the option to enter search text. You should include a filter condition that checks this text against the appropriate field, e.g. a customer name, site number, confirmation number, etc. This allows you to create any type of "Find" query you can think of.

Query Columns

This grid simply shows the "columns" of the Query (although the end use may not be columns, such as when used as a date header add-on Query for the Rack). The order of the entries here determines the order of the columns when the Query is used. While only the column heading name and expression is shown here, there is actually much more information in a column definition. Each column is actually a separate Query Column record, linked to the Query.

Typical functions are available for adding, inserting, copying, moving, and deleting columns. When adding or editing a column (double-click on a column also edits it), the Edit Query Column dialog is used to enter or edit the information for that column.

Quick-Add Fields

This is a special function for quickly adding simple data fields to a query. It displays a dialog where you can select one or more fields to be shown in the Query. You can choose fields from any table that will be appropriate, e.g. any table that will be available in the context of the query.

To add a field, just double-click the field name in the left column (or select the field and click the "-->" button). To remove a field, double-click it in the right-hand column, or select it and click the "<--" button. If you don't add them in the right order, don't worry about it here -- you can fix the order once you're done and get back to the Edit List Query dialog (using the Move functions).

For each field added, a Query Column record is created and added to the Query. The various details of the column, such as the heading, expression, justification, formatting, summing and sorting details are all set up with appropriate defaults. If these aren't quite what you want, you can edit the columns added, just like any other column.

Save & Test Query Results

This function invokes the Save & Test dialog, which shows the Query in a grid. This can be used to test the query, without completely exiting the Edit function. Be aware that it ***does*** completely save any changes you've made to the Query, so it negates any possibility of cancelling changes you've made.

Notes

You can enter any notes for yourself here, or use this as a description of the Query -- they're only seen here and in the list in Queries Setup.

Editing Query Columns

The Edit Query Column Definition dialog is only shown when you Add, Insert or Edit a column from the Edit List Query dialog. Here you can change any of the details of the query column.

Field / Expression

This expression defines the content of this column for each record of the query. You can enter the expression directly in this editing window, or use one of the helper functions above it. Buttons are available to Insert Expression Element and Test/Edit Expression, which invoke the corresponding dialogs to help build the expression. There's also an Insert Field button, which invokes the Select Data Field dialog. Here you can simply select a table and data field, and the expression and other formatting information will be automatically filled in with some appropriate defaults.

Column Heading

This will be the name of the column and the text in the heading. This field can't be blank, but it's OK to have duplicate names if you really want to. Keep in mind that the columns will auto-size to fit the widest thing in them, whether it's the heading or the data. So if the data is typically short, like a number, then you'll probably want to keep the heading short too.

Custom Colors for Column Heading

If this option is checked, you can select custom text and background colors for the heading. This isn't commonly used, but you might want to use it to highlight a column for some reason. An "Example" box will show the current colors, and the "Text" and "Background" buttons are used to pick the desired colors.

Data Color Scheme

If you want to use something other than the default Windows colors for the data in the column, then you need to use a Color Scheme. Appropriate context will be available in the color scheme, such as the record being shown and the from/to dates selected for the Query. Thus you can create a color scheme that colors the data according to content or anything else about the record being shown.

If you haven't created the color scheme you need yet, or if you find that adjustments to the color scheme are needed, then you can use the Edit button next to the color scheme selection list.

Tip: If you just want to use the normal Reservation color coding like the other tab views do, then create a Color Scheme that has "Reservations" as the Default scheme. No rules need to be added to the scheme.

Show Group Totals

If this box is checked, then the "totals" will be calculated for the column and shown at the bottom. If there are blank lines inserted in the query due to sorting groups (see "Blank lines..." below), then sub-totals will also be inserted in those blank lines.

The calculations for the totals shown can be a Sum, Average, or Count. Sum is obviously just the sum total, Average will divide the total by the number of records involved, and Count will simply show the number of records.

Tip: To get a total number of records at the bottom like you see on most tab views, check this box for the first query column and select "Count" as the totalling method.

Format

This determines the format of numeric values, assuming the expression results in a number. For non-numeric data, the "General" option should be chosen. For numeric data, select an appropriate format, e.g. Currency, Integer, Percent, or Floating Point. If none of these quite fits your needs, then you can select Custom Format and enter an expression to format the data any way you need to.

Note that technically you could format the data in the Field/Expression itself, for instance by using "Currency(Tran:Tran_Amount)" instead of just "Tran:Tran_Amount". However this would not allow the values to be totalled since the expression no longer results in a numeric value.

The Format specification also determines the format of the total and sub-totals, if the Show Group Totals option is selected.

Custom Format Expression

This field only appears when the Format selected above is "Custom Format". To edit the format, click the Edit button or simply click on the text box below it. This invokes the dialog to edit the format expression.

For the most part, the format expression should simply convert a numeric value to text. The numeric value to be formatted will be available with the context function **ThisValue**, which will already be shown in the expression the first time you edit it. There are several functions available for formatting numeric values. For a simple example, lets say that we want to take the number an show it as a currency value rounded to the nearest dollar. The expression entered would be:

```
Currency( Round( ThisValue(), 0))
```

Technically you can do anything you want in the format expression -- the context will also have the record of interest and the from/to dates of the query. So the format could even include other fields of the record.

Remember that the format only affects the way that the number is displayed, not the way it's sorted or totalled, which is why you would do the formatting here instead of in the main Field/Expression for the column.

Align Text & Heading

Select the desired alignment, or justification, of the data (text) and the column heading. Typically the heading is always centered, but there may be cases where you want it left or right justified.

Sort this column by default

When you want a specific column in the Query sorted when it's initially displayed, check this box for the column to be sorted. If no columns have this checked, then it will be sorted by the first column. You can also select the direction of the sorting, ascending (lowest on top, highest on the bottom), or descending.

Only one column in the Query should have this option checked (if more than one column does have it checked, the first column found will be sorted).

This does not affect the method used to sort the records -- it will be the same as if the column header is clicked to sort the column.

Blank lines between sorted groups

If this box is checked, and if the query is sorted by this column, then blank (non-record) lines will be inserted between any *different* values. This is similar to the Transactions detail view, where blank lines are inserted when it's sorted by date, type, category, etc. Only one blank line can be inserted between each different value, no matter how different they are.

Note: When deciding whether the values are different, **only** the result of the Field/Expression for the column is used. If a special sorting hierarchy is defined for this column, or a default sorting hierarchy for the Query is defined, it may affect the order of records within a "group" but it will not cause extra blank lines to be inserted.

When blank lines are inserted, any columns with the Show Group Totals option will also have sub-totals inserted in the blank line.

Pop-up Tip Expression

If you want something to appear in a pop-up hint whenever the mouse cursor is placed over this column, then you can enter an expression here. To edit the tip expression, click the Edit button or simply click on the text box below it. This invokes the Expression Creator dialog to edit the tip expression.

The result of the expression should be a text string to be displayed. Typical context will be available for the record underneath the cursor, so the expression can use fields from the record being shown, or any other information.

Note that the pop-up tip can only be a single line, so be careful about its length.

Double-Click Action Expression

This can be used to make something happen when this column is double-clicked on. To edit the expression, click the Edit button or simply click on the text box below it. This invokes the Expression Creator dialog to edit the expression.

This expression could be anything from showing a message box with information about the record to executing a Script performing all kinds of functions, even changing the values of fields. For instance, you might have a "Clean" flag defined for Sites, and make double-click change the flag back and forth between "Yes" and "No". Just be careful about doing things that can't be easily undone, since a double-click might be accidental.

Special Sorting Value/Hierarchy

This allows you to change the way this particular column is sorted, for instance when the header is clicked on to sort it. This is a list of expressions used for sorting, the same way the Query's Default Sorting Hierarchy works, but is only used when this column is sorted rather than for all columns. Any current sorting expressions are shown in a list. To edit the sorting expressions, click the Edit Special Sorting Value/Hierarchy button.

By default, the column will sort based on the result of the main Field/Expression -- so that result is compared directly for each record, whether it's a numeric, text, date, or boolean value. If this doesn't result in a reasonable sorting, then enter special sorting expressions to sort the way you want. If any sorting expressions are entered here, the Field/Expression for the column will not be used at all for sorting.

The most common value needing special sorting is the Site name or abbreviation. Sorting by the text of the site name will rarely result in the order you want. In this case, you actually want to sort by their record order, that is the order they have been set up in the Sites data table. Therefore you would use this expression for sorting a Site column:

```
SiteOrder( Site() )
```

Another common issue is upper-case and lower-case values, since "Smith", "SMITH" and "smith" would all be considered different when compared directly. For this you would want to convert them to all upper-case and sort that way so they're all considered the same, without actually changing the data shown in the column. This would be done using the Upper() function -- see an example in the next section, Sorting Hierarchies.

Sorting Hierarchies

The Edit Sorting Hierarchy dialog is used to enter one or more expressions to determine a sorting order of records, for instance in Queries. Each expression is shown in a grid, with the typical functions to Add, Edit, Copy, Delete and Move the expressions.

Since the expressions are used in the order shown here, the order of expressions in the list determines the sorting hierarchy. In other words, the first expression is the most important and is compared first, and if that still results in equal values then it will compare using the next expression, etc.

There will be appropriate context information for each expression, since the expressions are used to compare specific records or other specific information to be sorted. The exact nature of the context will depend on where the sorting hierarchy is used, but for instance if it's the default sorting hierarchy in a Query with a base table of Customers, then the context function **ThisCust** will be available (or the generic version, **ThisRecord**, can also be used).

When sorting is done, it actually executes each expression twice (once with the context of each record to be compared), and then compares the results. Therefore the expressions can result in any type of value -- text, numeric, date, etc. -- the program will know how to compare them. You don't have to actually do the comparison in the expression, just make the expression be the values to be compared.

Lets say we have a Query with the Reservations base table. To have a default sorting of customers by last name, and then secondarily sort by first name if the last names are the same, and a third level of sorting by the number of nights in the reservation (if the same customer), the sorting expressions would be:

```
Upper( Cust:Cust_Last_Name )
Upper( Cust:Cust_First_Names )
Resv Nights( ThisResv() )
```

Note that we used the Upper() function on the names so that it doesn't matter whether the names are upper or lower case (or mixed-case). Also note that the shorthand field descriptor is used (e.g. Cust:Cust_Last_Name) instead of a function like FieldText(ThisCust(), "Cust_Last_Name"). The shorthand should generally be used everywhere possible, since it does more work during parsing and less during execution

Filter Conditions

The Edit Filter Conditions dialog is used to enter one or more filtering expressions for Queries. Each expression is shown in a grid, with the typical functions to Add, Edit, Copy, Delete and Move the expressions. Since the expressions are used in the order shown here, this can be a speed consideration but otherwise is not an issue (see below).

Filtering conditions determine which records of a Query's Base Table are included in the Query. Without any filtering conditions, ALL records are included (which in some cases is what you want, e.g. to include all Sites).

Each filter condition is an expression that results in a boolean value (True or False). It's important to remember that **all** of the conditions must be "met", or "True", for a given record to be included in the Query. When filtering a record for the Query, each expression is executed in the order specified, until a False condition is found. If no condition returns False, then the record passes the test and it's included in the Query. If a False is found, it stops executing filter expressions (to save time) and the record is not included.

The most common elements of the filter conditions are the context functions for the data range that the user selects for the Query (e.g. the From and To dates on most Tab views). For instance, in a Reservation Query you can include only Reservations that exist in the date range with the following filter condition:

```
Resv:Resv_Last_Date >= FromDate() AND Resv:Resv_First_Date <= ToDate()
```

This is just like the On Site view date filtering. By changing the Last or First date fields, you can change it to include reservations arriving or departing on the selected dates. Similar filtering would be done for Transactions using the **Tran:Tran_Date** field descriptor.

Other common filter conditions would be for including only active reservations (using the **ResvIsActive** function), including only checked-in reservations (using the **Resv:Resv_Status** field descriptor), or including only reservations on a site (using the expression **Site() != NullRecord()**).

Text Search Conditions

If the Query includes a Text filter, then you need to include a condition for that. The text to be searched (entered by the user with the F9 key or Search button) is available in the context function **ThisSearchText**. Here's an example filter expression to look up transactions that include the entered text in their receipt number:

```
ThisSearchText() != "" AND Find(Tran:Tran_Invoice,ThisSearchText()) != 0
```

Note that the expression above also checks for the search text being blank, so that if no text is entered then it will result in False -- no records will pass the test. That keeps the Query from showing all records until search text is entered (which could be rather slow if this is the only condition).

Speed Considerations

Note that any number of filter condition expressions can be used. Technically, you could include all filter conditions in one expression (using "AND" logic), but this can actually result in a slower query because it has to execute every part of the expression. If the filtering is broken up into multiple expressions, then it only executes the expressions until a False condition is found and then it can skip the rest.

If speed is a significant issue, you can rearrange the conditions to put the most likely "False" conditions at the top (so less expressions are executed for most records). Or if some expressions are more complicated than others, you can put the "faster" expressions at the top and leave any really slow ones for last, which would only executed if the record passes all other "quick" tests.

Also note that all filter expressions are pre-parsed -- that is, the parsing portion of the expression processing is done only once each time the Query is refreshed, so it doesn't have to be done every time the expressions are used to filter a record.

Filter Expression Errors

It's important to know that if a filter expression results in an error, then the record is considered to have passed the test (the same as the condition resulting in True). The reason for this is that it's easier to locate the problem if it shows too many records than if it shows none at all.

Save & Test Query

This function is invoked from the Edit dialogs for either List Queries or Cross-Table Queries. This dialog may also be shown for a particular Query through the **ShowQuery** Expression function. It simply shows the Query in a grid, with From and To date selections like the Queries Tab View would. If the Query has a text filter, you can also enter the text to filter. This allows quickly testing the query to see what it will look like, without completely exiting the Edit function.

Note: Be aware that clicking Save & Test from an Edit Query dialog ***does*** completely save any changes you've made to the Query, so it negates any possibility of cancelling changes you've made.

When you use the Save & Test function from an Edit Query dialog, any errors in the expressions of the query will be shown here as pop-up tips on the cells (when you put the mouse over the cell). Sometimes an error will mean that a cell appears blank when it should otherwise have data, so try holding the mouse over any blank cells to see if there's an error message. Note that errors also appear this way when the Query is shown elsewhere also, but only if you're logged in as an Administrator.

This dialog simulates the Queries Tab View in nearly all respects, including the right-click menu functions and double-click actions.

Editing Cross-Table Queries

The Edit Cross-Table Query Definition dialog is shown when adding or editing cross-table queries from Queries Setup. Other functions where Queries are referenced, such as the Queries Tab view, may also have a button to directly Edit the Query without leaving that function and going through Queries Setup.

Here you can edit all of the components of the Query. The top portion has a few basic fields you can edit directly and buttons for editing the Filter Conditions for the Query and Testing the Query. Next is a grid for editing the "Axis/Grouping" definitions, and the main portion of the dialog for editing the "meat" of the Query -- the Data Expression and various formatting/action information.

Query Name

The name should be descriptive enough for selecting the Query out of a drop-down selection list. Queries will usually be shown in the order they appear in Queries Setup, not alphabetical, so the name doesn't affect the order. Each Query must have a unique name (which is not case-sensitive).

Base Table

The base table determines the primary data table of the Query -- that is, which records are potentially going to be included in the Query. For instance, if the base table is Reservations, then the Query will include all Reservation records by default, subject to the Filtering Conditions.

Any table in the database may be selected, but only a few are useful for most situations -- Reservations, Customers, Transactions, and Sites.

For more details about base table selection, refer to Editing List Queries in a previous section.

Save & Test Query Results

This function invokes the Save & Test dialog, which shows the Query in a grid. This can be used to test the query, without completely exiting the Edit function. Be aware that it ***does*** completely save any changes you've made to the Query, so it negates any possibility of cancelling changes you've made.

Access Level

The access level simply determines which operator access level is required to view the Query. If the current operator does not have the selected access level, then the Query will not be shown in the selection list on the Queries Tab view, for instance.

Exclude from Lists

When this is checked, the Query will not be shown in any selection lists such as in the Queries Tab view. Note that this does not actually disable the Query -- e.g. if it was already selected as an add-on for a tab view then it will still be used in that function. However it won't be available as a general selection, which can be handy for keeping special-purpose Queries out of view. Of course this also means that you would need to uncheck this again (temporarily at least) if you ever did need to select the Query somewhere. Another alternative would be to set the Access Level for such Queries to Administrator, assuming that most operators are non-administrator.

Note that one place this does not affect is the Tab Views Setup (since this is assumed to be an administrator-only function anyway).

Filtering Conditions

The filtering conditions determine which records of the Base Table are included in the Query. Without any filtering conditions, ALL records are included. Refer to the Filter Conditions section above for more details.

Includes Text Filter

When this option is checked, the user will have the option to enter search text. You should include a filter condition that checks this text against the appropriate field, e.g. a customer name, site number, confirmation number, etc. While not necessarily useful for most cross-table queries, you can use it as a free-form filter, e.g. to show the results for only a certain operator, or only a certain reservation type (assuming you have the Filter Conditions set up accordingly).

Axis/Grouping Definitions

The Axis/Grouping definitions determine what kind of cross-correlation the query is going to show. These are shown in a grid, and the Add/Edit/Delete buttons to the left are used to modify them as needed. Note that the order of these in the grid does not matter. Currently there must be exactly two axis definitions, one for rows and one for columns, although the grid is designed to hold more than two (for future expansion, e.g. to allow sub-totalled cross-tables).

The most common combination of groupings uses one axis for dates (e.g. daily or monthly totals) and another for some other grouping of interest (e.g. transaction categories, reservation types, site types, etc.) This equates to the "Group by" and "Summ by" selections in transaction summary reports and most statistical reports available in the Reports menu. However it's also possible to make both groupings non-date, e.g. to cross-correlate reservation type with discount type, or how-heard with rig type (use your imagination!).

For more details, refer to Cross-Table Axis/Groupings.

Calculated Data Expression

This defines the "meat" of the Query. To edit the expression, click the Edit button or simply click on the text box below it. This invokes the Expression Creator dialog to edit the data expression.

After all of the cross-axis grouping is done to figure out what records are to be used to calculate each cell of the Query, the Data Expression is executed for each cell (each row and column combination). Obviously you need a way to get the list of records in the cell, and a way to do calculations from those records. The context functions **ThisListCount** and **ThisListRec** are used to access the records for each cell. Most cross-table Queries will involve summing up something about the records, so here's a simple example to add up the amount of Transactions for each cell (assuming the base table is Transactions):

```
LoopSum(1, ThisListCount(), "#i#", 'TranBalAmount( ThisListRec( #i# ))')
```

The expression above simply sums up the transaction amounts for each record in the list (ThisListRec) from 1 to the number of records in the list (ThisListCount).

There may be cases where you don't need to do anything with the records in the list, just show how many are in the list (included in the group), like in an Arrivals Statistics report. In that case the data expression can simply be **ThisListCount()**, which returns the number of records in the grouping.

While the expression's results don't need to be numeric, keep in mind that if you plan to show any kind of totals for the rows or columns then it must be numeric.

Data Color Scheme

If you want to use something other than the default Windows colors for the data in the grid, then you need to use a Color Scheme. Appropriate context will be available in the color scheme, such as the list of records being shown for the cell and the from/to dates selected for the Query. Thus you can create a color scheme that colors the data according to content or anything else about the record being shown. There are other context functions available specifically for cross-tables, which usually start with "ThisGroup". For instance, **ThisGroupText** will contain the heading text of the cell's group, and **ThisGroupFromDate** will contain the starting date for the cell.

The most common use for this in Cross-table Queries is to show negative values in red. This can be done easily using the **ThisValue** context function to get the value being shown in the cell. For example, create a Color Scheme with this expression for rule with red text:

```
ThisValue() < 0
```

If you haven't created the color scheme you need yet, or if you find that adjustments to the color scheme are needed, then you can use the Edit button next to the color scheme selection list.

Format

This determines the format of all numeric values in the Query, assuming the Data Expression results in a number. For non-numeric data, the "General" option should be chosen. For numeric data, select an appropriate format, e.g. Currency, Integer, Percent, or Floating Point. If none of these quite fits your needs, then you can select Custom Format and enter an expression to format the data any way you need to.

The Format specification also determines the format of the row and column totals, if they are used in the Query.

Custom Format Expression

This field only appears when the Format selected above is "Custom Format". To edit the format, click the Edit button or simply click on the text box below it. This invokes the Expression Creator dialog to edit the format expression.

For the most part, the format expression should simply convert a numeric value to text. The numeric value to be formatted will be available with the context function **ThisValue**, which will already be shown in the expression the first time you edit it. There are several functions available for formatting numeric values. For a simple example, lets say that we want to take the number an show it as a currency value rounded to the nearest dollar. The expression entered would be:

```
Currency( Round( ThisValue(), 0))
```

Align Text

Select the desired alignment, or justification, of the data in the query grid (including any totals shown).

Pop-up Tip Expression

If you want something to appear in a pop-up hint whenever the mouse cursor is placed over a cell, then you can enter an expression here. To edit the tip expression, click the Edit button (or simply click on the text box below it). This invokes the Expression Creator dialog to edit the tip expression.

The result of the expression should be a text string to be displayed. Typical context will be available for the cell underneath the cursor, so the expression can use the value in the cell, group information, or fields from the list of records being shown.

Note that the pop-up tip can only be a single line, so be careful about its length.

Double-Click Action Expression

This can be used to make something happen when a cell is double-clicked on. To edit the expression, click the Edit button or simply click on the text box below it. This invokes the Expression Creator dialog to edit the expression.

This expression could be anything from showing a message box with information about the cell's contents to executing a Script performing all kinds of functions, even changing the values of fields. Just be careful about doing things that can't be easily undone, since a double-click might be accidental.

Notes

You can enter any notes for yourself here, or use this as a description of the Query -- they're only seen here and in the list in Queries Setup.

Cross-Table Axis/Groupings

The Edit Cross-Table Query Axis/Grouping Definition dialog is only shown when you Add or Edit an axis/grouping from the Edit Cross-Table Query dialog. Here you can define the grouping criteria for a row or column of the Query.

The term "Axis" comes from mathematical X/Y graphing, were one axis is horizontal and one is vertical (in this case, rows and columns of values in a grid). From here on, we'll just refer to them as "Groupings", since in the context of this dialog you're really defining how the records are grouped to form rows or columns.

A Cross-table Query must have a Row grouping and a Column grouping, creating a cross-table grid of values where each cell in the grid results from records that meet both the row and column condition for that cell. For instance, lets say you have a Reservation-based cross table, with the column grouping being months and the row grouping being Reservation Type. So for each month you'll have a column, and for each Reservation type you'll have a row, and each cross-table cell contains the records meeting the month and type conditions for that column and row.

Note that whenever we refer to "base table records" or "all records" here, this is of course subject to the Filter Conditions of the Query itself.

So with that in mind, remember that this dialog only defines ***one*** of the two groupings at a time, either the rows or the columns. You'll be adding each of the groupings from the Edit Cross-Table Query dialog.

Axis name

This is just a name for your reference, which will be displayed on the Edit Cross-Table Query dialog. Keep it simple but relevant, like "Dates" or "Types".

Axis Type

This is either "Rows" or "Columns", whichever this grouping defines. Later versions may allow other options for sub-groupings.

Grouping Type

There are several types of default groupings available, which mostly just help you define the conditions of the query more easily than raw expressions alone, but also may affect what context information is available for the group's expressions.

Each type of grouping requires different information to define it, as described below.

- None (single result) -- If you only want a single row or column, without any grouping.
- Dates -- Group by date, e.g. daily, monthly, etc.
- Records of a Table -- Create a group for each record in a selected table, e.g. each Site.
- Items of a Pick List -- Create a group for each item in a selected Pick List, e.g. each Site Type.
- Items of a Fixed List -- Create a group for each item in a selected Fixed List, e.g. each Transaction Type.
- Expression, Filtered or All -- Create groups with headings according to an expression you define (see below for details).
- Custom Groupings only -- Don't create any default groups, but use the ones you defined.

When a grouping type is selected, you'll see two things happen -- Other fields may appear for entering the details, and a default Group Conditional Expression is supplied to help you get started (which in some cases is already exactly what you need).

Date Grouping

If the "Dates" grouping type is selected, you can select a Date Grouping of Daily, Monthly, Quarterly or Yearly. The date grouping of course defines what groups are created (limited to the From and To date range in effect for the Query), and the date range that will be included in each group. When any date grouping is used, the context functions **ThisGroupFromDate** and **ThisGroupToDate** can be used to get the date range included in the group, in any of the group expressions.

You can optionally provide a Group Headings Conditional Expression. This expression will be executed for each date range being considered as a group, so that you can filter which groups in the overall date range will be included in the report (for instance you could use this to only include certain days of the week). The context functions **ThisGroupFromDate** and **ThisGroupToDate** will be available for this conditional expression.

Note that no matter what date grouping type is used, the From and To dates selected for the query will limit the actual groups created (and will most likely limit the records included in the query, since they're most likely being filtered by date in the Query's Filter Conditions.) Thus a Monthly column grouping will show a month for each column, but the data contained in the column may still only include one day.

Table & Heading Field

When "Records of a table" grouping is selected, you need to select the Table to be used and the Heading Field to be used for the group headings (row or column headings). For instance you might select the "Sites" table and the "Site Name" for the heading field so it shows the site names. A group will be created for each record of the selected table, regardless of any filtering -- ***every*** record of the table considered a group even if none of the Base Table records are related to it (e.g. even if a Site is inactive or has no reservations, it would be included). The default sorting of the records is the actual record order in the table.

The context function **ThisGroupRec** will be available to get the group's record (e.g. the Site for this row) in the group expressions, and **ThisGroupText** can be used to get the fields text (the group heading).

Pick List or Fixed List

When the "Items of a Pick List" or "Items of a Fixed List" grouping type is selected, you need to select the list to be used. A group will be created for each pick list or fixed list item (subject to the optional conditional expression as described below). For fixed lists, the headings will be the common item selection text. For Pick Lists, the Selection name field of the pick list items will be used. The default sorting of the groups is the order of items in the pick list or fixed list. The context function **ThisGroupText** can be used to get the list item's text (the group heading).

Group Headings Conditional Expression (for Date, Record, Pick List and Fixed List groupings)

This optional expression allows you to filter which groups are actually included in the report. For instance in a Records of a Table grouping, a group would normally be included for every record (e.g. every Site). By specifying a conditional expression here, you can limit the report to only show certain records (e.g. filter out by site type, park, etc.).

This expression is executed for each potential grouping, e.g. each record, date group, or list item. As with any Conditional expression, it should have a boolean (True/False) result -- if the result is True, then the group (record or item) will be included. The context available depends on the grouping type -- for Records of a Table use **ThisRecord()**, for Items of a Pick List or Items of a Fixed List use **ThisGroupText()**, and for Date groupings use **ThisGroupFromDate** and **ThisGroupToDate.**

Group Headings Expression (for "Expression" grouping types)

When "Expression, Filtered" or "Expression, All" is selected for the grouping type, you also need to enter a Group Headings Expression. This expression will be executed for each record of the Query's base table (**ThisRecord** will have the record context) and must have a text result type (within this expression. Each unique text result created will be used as a group definition, the result text being used for the actual row or column headings. The context function **ThisGroupText** can be used to get the expression's text (the group heading). By default these headings are sorted in the order that the unique values are found (e.g. no sorting is done).

If "Expression, Filtered" is selected then only records that pass the Query's Filter Conditions will be used to determine the groups (i.e. the Group Headings expression is only executed in the context of those filtered records, and only those results will be used as groups). If "Expression, All" is used, then the filtering is ignored and all records of the base table are used.

A common use of this type of grouping is to group transactions by Operator, since the Operator field of transactions is just a text field. if the "Expressions, Filtered" grouping type is used, then you would expect the report to only include operators involved in at least one transaction in the date range of the report. If the "Expressions, All" grouping type is used, then the report will include all operators who were ever involved in a transaction, even if they didn't have any for the transactions in the Query's date range. Either way, the Group Headings Expression in this example would simply be **Tran:Tran_Oper**, which returns the operator name for the transaction. Of course this assumes that the base table for the Query is "Transactions".

Show totals for each group

If this box is checked, then the "totals" will be calculated for the column or row and shown at the bottom or side. If the Axis Type is "Columns", this determines whether totals are shown at the bottom (totalling each column). Likewise, if the Axis Type is "Rows" then this option will show totals on the right-hand side, totalling each row.

The calculations for the totals shown can be a Sum, Average, or Count. Sum is obviously just the sum total, Average will divide the total by the number of records involved, and Count will simply show the number of records.

Align headings text

This is simply the text justification used for the headings. Usually "Center" is best for columns and "Left" for rows.

Group headings color scheme

If you want to use something other than the default Windows heading colors, then you need to use a Color Scheme. Appropriate context will be available in the color scheme for the grouping (as described above), in addition to **ThisListCount** and **ThisListRec** for access to all records included in each group. Thus you can create a color scheme that colors the data according to group's heading, record or date information, or anything about the records being shown (e.g. the color could be based on how many records are included in the group or even the total transactions amount).

If you haven't created the color scheme you need yet, or if you find that adjustments to the color scheme are needed, then you can use the Edit button next to the color scheme selection list.

Group Conditional Expression

This is the most important aspect of the grouping -- The Group Conditional Expression determines which records (from the Query's Base Table) are to be included in each group. The expression is executed for every record to be included in the Query (subject to the Filter Conditions for the Query), with the context of each group (the row or column grouping information, from context functions as described above). The expression must return a boolean value -- True if the record should be included in the group and False if it should not.

Note that the expression is executed for every grouping expression (for each record) to see if it's included in the group, not just until it finds a True result -- thus it's possible that a record might be included in more than one group, or even all groups. This is normally not desired, but it's certainly possible with the appropriate expressions.

When the Grouping Type is selected, a basic default expression appropriate for that grouping is entered here automatically. In some cases where it can't determine what needs to be used, it will include text like "<fill in field name>". You just need to edit the expression and insert the appropriate field name. Of course these are just basic assumptions, and may not be what you need, but it's intended to provide a hint of what's needed in the expression.

Group Heading Sorting Hierarchy

If the order of the groups is not what you want by default, then you can use one or more expressions to determine the group order in the rows or columns. The context of the group is available for the sorting expressions as described above (e.g. the heading text, record or date information). To edit the sorting hierarchy, click the "Edit group heading sorting hierarchy" button above the list.

Note that this sorting is only done for the "default" groupings as defined by the Grouping Type. It does not affect any Custom group Definitions -- all custom groupings will appear after the sorted default groupings.

Context for Expressions

The context functions available for each grouping type are detailed below. These context functions are available to any expression executed for a particular cell in the cross-table:

- The Group Conditional Expression
  ThisRecord() / ThisResv() / etc. for the record being testing for inclusion in the group
  ThisGroupRec() if it's by table,
  ThisGroupText() if it's a records of a table, pick list, fixed list, or expression
  ThisGroupFromDate() / ThisGroupToDate() specify the group's range for a date grouping

- The Group Heading Sorting Hierarchy expressions
  ThisGroupRec() if it's by table,
  ThisGroupText() if it's records of a table, pick list, fixed list, or expression
  ThisGroupToDate & ThisGroupFromDate if it's grouped by dates

- The Group headings Color Scheme
  ThisListCount() & ThisListRec(n) of the records
  ThisGroupRec() if it's by table
  ThisGroupText() if it's a table, pick list, fixed list, or expression
  ThisGroupToDate & ThisGroupFromDate if it's grouped by dates

- The Calculated Data expression (in the Edit Cross-Table Query dialog)
  ThisListCount() & ThisListRec(n) of the records
  ThisGroupRec(), ThisGroupText() for the non-date axis
  ThisGroupFromDate() & ThisGroupToDate() for the date axis

- The Data Color Scheme (in the Edit Cross-Table Query dialog)
  ThisListCount() & ThisListRec(n) of the records
  ThisValue() & ThisTextValue() for the body results & totals
  ThisGroupRec(), ThisGroupText() for the non-date axis
  ThisGroupFromDate() & ThisGroupToDate() for the date axis

- The Pop-up Tip expression (in the Edit Cross-Table Query dialog)
  ThisListCount() & ThisListRec(n) of the records
  ThisGroupRec(), ThisGroupText() for the non-date axis
  ThisGroupFromDate() & ThisGroupToDate() for the date axis

- The Double-Click Action expression (in the Edit Cross-Table Query dialog)
  ThisListCount() & ThisListRec(n) of the records
  ThisGroupRec(), ThisGroupText() for the non-date axis
  ThisGroupFromDate() & ThisGroupToDate() for the date axis

Note that there's only one set of context functions for date groups and one for non-date groups (text or record). This assumes that one grouping will be by date and the other will be something else. If you have a cross-table where both groupings are by date or both groupings are non-date, then the context functions will only contain information for one of the groupings (either rows or columns). It's somewhat random which grouping's information is available in this case and it could change in future versions, so it's best not to depend on it being either one.

Cross-Table Custom Groupings

The Edit Custom Grouping Definition dialog is invoked from the Add or Edit Custom Group button in the Edit Cross-Table Axis/Groupings dialog. This dialog has a Group Heading field (the row or column heading text) and a large window to edit the Grouping Expression. It also has buttons to Insert Elements and Test/Edit the expression, which work like the Expression Creator dialog.

You can add any number of custom group definitions to a cross-table grouping definition. Each one is simply a column heading and an expression. A list of any already defined is shown on the Cross-Table Query editing dialog with the typical buttons for adding, editing, moving, copying and deleting them. The order of the definitions in the list determines the order in the Query, and any defined as custom groups will appear "after" any default groups as defined by the Grouping Type -- e.g. they will be the bottom-most rows or the right-most columns (not including Totals). Of course if "Custom Groups only" is selected as the grouping type, then only these custom groups will be shown in the Query.

A typical use of custom groups is to include a "(none)" group when the normal grouping is by Sites, How Heard, Discount, etc. where there might be records with no value for this information. For instance to include a no-site grouping, add a custom grouping with the expression **Site() = NullRecord()**. They also allow adding special columns to transaction reports like Previous and All-Time, or create any type of special groupings you can come up with, providing complete flexibility in the creation of summary reports.

The custom group expressions should have a boolean result, just like the Group Conditional Expressions -- that is, the expression should return True if the record should be included in the custom group, and False if not. Each custom group expression is executed for each record in the Query's base table, just like the Group Conditional Expressions. However the only context information available besides the general query information is the base table record (**ThisRecord**) and the custom group's heading text.

Forms

Overview

The Forms functionality in Campground Master encompasses nearly all types of printable formats that aren't grid types of reports. All customer receipts, envelopes, mailing labels, window tags, purchase orders and even E-mail message formats are part of the Forms definitions. Note that we define Forms only as printable output formats, not as user input formats the way "Forms" are defined in Microsoft Excel or Visual Basic. We refer to input forms as Dialogs, which are set up in a separate section.

Previous versions of Campground Master had a set of receipts, etc. available as "canned" or pre-programmed formats with a few formatting or content options. These are all still available as a pre-configured set of default Forms, so that upgrading will be seamless and new installations have a base set of common formats to start with. These use the same internal code to ensure that no change will be noticed in most cases (there are a few enhancements, such as multi-page receipt support, which affect the "old" receipt formats). All of the previous settings from Printing Options, Notice text, etc. are still used and they still affect these default formats (and in some cases will affect customized versions of these as well).

The Forms Setup functionality adds several levels of customizability not previously available (each "level" of customization below requires more technical knowledge):

1. Any of the default receipts can be disabled, renamed, or rearranged in the selection lists to suit your preferences.
2. Form elements such as text, data or graphics can be added on top of existing "canned" receipt formats.
3. A format can be modified more extensively by importing custom Form templates, which replicate the canned Forms in most respects, and making changes to them.
4. Completely custom Forms can be created from templates or from scratch.

Any modifications other than the first level above will require at least some knowledge of Expressions, and a fair amount of programming expertise is recommended for the 3rd and 4th levels.

In addition, a special "Section" feature allows creation of multiple-page Forms with different information on each page. For instance you could create a single 4-page "receipt" format which includes 2 copies of the Ticket Form (one requiring a signature for your records), a Window Tag, and a letter explaining all of your policies (however all sections in a single Form must go to the same printer).

Forms Setup

To create a Form, go to Maintenance / Advanced Customizations / Forms. This opens the Forms Setup dialog, which lists all current Forms and has the typical functions for Adding, Inserting, Editing, etc. Note that while there are functions to Move Up and Move Down, a Form's position in the list does not affect any functionality other than its order in drop-down Form selection lists that the user sees. This of course may be important to you for organizational purposes.

You can also Export one or more Forms to a text file, or Import Forms. This is primarily for you to import Forms created by the software provider, though it can also be used to transfer Forms between multiple databases.

Forms cannot have duplicate names (or else they could not be uniquely selected from a list). If you make a Copy of a Form, text like "(copy 1)" will be added to the name to make sure it's unique. Of course you can change this to be more appropriate. Duplicate checking for the names is not case-sensitive ("My Form" is considered the same as "my form").

Add the default receipts

This special function will add all of the default receipts to the list (plus other Forms like envelopes and labels). The main purpose of this is in case you make some changes and then want to get back the original settings. Note that it will only add defaults that do not already exist, according to the Form names, since duplicate names are not allowed. Thus if you make changes to a default that you want to undo, you need to delete the original first before adding the defaults. A "fresh" copy of any deleted will then be added to the end.

Renaming Forms

If you want to change the name of a Form (for the selection lists), then select the Form in the list and click Edit Form definition. The Edit Form dialog will show the current Form name at the top. Just change the name as needed and Save it.

Rearranging Forms

The order in which the Forms appear in selection lists, for instance in the Transactions dialog, is determined by the order that they appear here in Forms Setup. If you want to move them around, for instance to make the most-used Forms appear at the top (or nearer to the default Form selected in Printing Options), then use the Move Up and Move Down buttons to move one or more selected Forms.

Disabling Forms or changing access levels

If you don't want so many Forms in the receipt selection lists, you can disable the ones you don't use. Technically you could also Delete them, but it's better to just disable them in case you want to use them later. To disable a Form, select it in the list and click Edit Form definition. Then in the Edit Form dialog, uncheck the "Enabled" box and then click Save.

Changing the access level for a Form is basically done the same way -- go into Edit Form Definition, and select the desired Access Level.

Changing the number of transactions printed on a page

The space available for transactions on the receipts can vary depending on the number of "Additional sites" (linked reservations) or Receipt #'s (separate transaction sessions), as well as other factors like including credit card information. Each Form is set up with a specific maximum number of transactions that it can print per page (it's not "intelligent" enough to figure that out automatically for each receipt), but this may be too many in some cases, or perhaps you find that it can fit more. To adjust this, select the Form in the list and click Edit Form definition. Then in the Edit Form dialog, change the "Trans rows/page" value and then click Save.

Adding elements to the default formats

Each of the default format entries is set up as an "Add-on" Form, essentially as a blank custom Form on top of the corresponding canned default Form. This means that it still uses internal code to create the receipt Form, instead of using custom Form elements, so that all Forms still work the way they did in previous versions. These "canned" Forms are also a little more flexible than a custom Form could be, and also faster.

However since they are still Form definitions, you can add Form Elements to them. Anything you add will be printed "on top" of the Form selected in the "Add-on" field. Thus you can add a logo image, some extra text, or data fields to otherwise blank areas of a Form. (Technically you could overwrite existing Form information, but this is very tricky.) To do this, just select the Form and Edit it, and add Form Elements as required. Just keep in mind that the position of fields on most canned Forms isn't fixed -- much of the information will move around depending on what information is actually needed for the given reservation or customer.

Making other changes to the default formats -- Importing and changing templates

As mentioned above, the default formats are just Add-ons to the canned Forms, which don't allow changes to the Form other than additional elements. If you need to make changes to the Form, such as moving, renaming or deleting elements, or adding things to it that won't have a fixed position, then you need to be able to edit the actual Form elements. This isn't possible with the canned Forms, so we've created sample templates that duplicate the canned formats as close as possible. There are a few aspects that aren't quite possible to duplicate (or were too difficult to be worthwhile), but for most users this won't be a problem.

To use these templates, you must first Import them. Click the Import Form(s) button, and you'll get a typical Windows file dialog labelled "Import Forms". You need to locate the sample Forms folder, which is typically

C:\Program Files\Campground Master\Samples (most likely you just need to double-click the "Samples" folder to get there). Now select the appropriate file, for instance "Sample Form - Ticket Form" to get the Ticket Form template, and Open.

Note that the import/export files use the "CSV" file extension (e.g. Sample.csv), which means it's a comma-separated-value text file. Windows may recognize this file extension as something another program can open like Excel, but these are in a special format for importing records to Campground Master and should not be used in other functions. Also avoid opening different kinds of samples which use the same extension (e.g. don't open a Form sample from an Import Script function).

When you're importing sample Forms, it may also import Macros or Scripts that are used in the Forms. If these are already defined, resulting in a duplicate name, then a warning will be shown listing the duplicates and what their names were changed to during the import. These might be safe to delete, assuming the imported version does the same thing as the original version. Otherwise you will need to change any expressions in the Forms that use the Macro or Script so that it uses the correct name.

Once the sample is imported, you'll see it appear in the Forms list (probably with a name starting with "Custom". You'll probably want to move it to the top of the list for easier location. Now you just need to Edit the Form to make any changes you need. You'll notice that the sample Forms make heavy use of Regions to sub-divide the Form's data areas. This is recommended so that you can do things like export/import regions, move entire areas easily, and "name" regions for easy reference when editing.

Note that these sample templates do contain the entire Form defined as Form Elements, so any aspect of the Form can be changed. Even if you want to create your own Form that's nothing like one of the others, for instance a letter that's nothing like the normal confirmation letter, you might as well import the confirmation letter template and delete everything except the header regions (the park & customer address information) so you have a good starting point.

You may also notice that the sample Forms use the settings from Printing Options and Park Setup wherever it's applicable, e.g. to show the site name, abbreviation or type, and to get the text to be shown at the bottom of receipts. This is done by using functions like **SettingText()** or **SettingLocalBool()** in the Data or Condition expressions of the Form elements. If you're making your own custom versions, you may want to replace these with your own text or conditions as appropriate, or you may prefer to leave them so that changes to Printing Options still affect your custom Form.

Creating custom Forms from scratch

If none of the sample modification options above works for you, then of course you can create a Form from scratch. Just click Add Form or Insert Form, enter the name and basic information and start adding Form elements. See the following Editing Forms section for details.

Editing Forms

The Edit Form Definition dialog is shown when adding or editing Forms from Forms Setup. Other functions where Forms are referenced, such as the Form Selection dialog, may also have a button to directly edit the Form without leaving that function and going through Forms Setup.

Here you can edit all of the components of a Form. There are a few fields you edit directly here, and the main portion of the dialog for editing the "meat" of the Form -- the Form Elements.

Form Name

The name should be descriptive enough for selecting the Form out of a drop-down selection list. Forms will usually be shown in the order they appear in Forms Setup, not alphabetically, so the name doesn't affect the order. Each Form must have a unique name (which is not case-sensitive).

Enabled

The Form can be disabled so that it does not appear in receipt selection lists (just uncheck this option). Technically you could also delete the Form, but it's better to just disable it in case you want to use it later. This is also handy to disable Forms you haven't finished, so the users don't try to use it (giving it a high Access Level may also be helpful for this).

Form Type

There are several Form types to choose from, which primarily determine where the Form will be shown as a selection. For instance a Reservation Receipt format won't be shown if printing a receipt for Unbound Transactions, and Label Forms will only be shown where it's possible to print a labels (where multiple records are involved, e.g. the Reports / Mailing Labels function or Find Customers).

Besides being useful for keeping Forms organized according to their use, some types affect what the Form can contain or what other settings it will have. For instance, only the Receipts Form types can have transactions in them (at least for the normal purposes of showing them in a Transaction Table element). Other special situations are mentioned below.

Labels print multiple "Forms" per page (e.g. one instance for each record in a list), and thus have settings for position, spacing, and the number of rows and columns per page. Of course this technically doesn't have to be just mailing labels -- you could use this for any special Forms where several records need to be printed per page, such as a reservation summary report with many details per reservation that wouldn't fit in a single-row-based Query.

E-mail Form types must have "Text output" for the Printer selection (they can't actually be printed), and "Character positions" for the Format (their elements can't be positioned in absolute coordinates on a page).

The Window Tag type has a special situation -- if the "Print Window Tag" button is used on the Reservation Transactions dialog, it will auto-print the top-most Form with this type. This is a special case where the order of Forms is also important. Note also that Window Tag Forms should always be Reservation-based.

Add-on

If this is checked, you can select one of the "canned" formats as a base for the Form. Nothing about this canned Form can be modified, but you can add elements on top of it. This is a means to make simple add-on changes to one of the canned Forms, and of course it's also how all of the default Forms are set up (ass add-ons with no added elements). Be careful to select an appropriate Form Type for the add-on selected, or else the results may be unpredictable.

If the format of the Form is Character-positions, such as a 3" receipt or E-mail, be aware any elements are added on at the end of the canned Form, not in a specific positions.

Format

The format of a Form determines how things are positioned on it. There are basically two choices -- "Variable positions" (in inches or millimeter units), and "Character positions". Note that the inches / millimeter choices for variable can be changed any time, it won't affect the Form (all Forms are actually stored with inches, and conversions are done only for your convenience in the editing dialogs).

Any Form that's used with a Windows printer driver, e.g. for an ink jet or laser printer, can use the Variable positions format. This allows you to place Form Elements anywhere on a page based using absolute positions, and also allows you to use "Region" elements to sub-divide the Form definition.

The Character position format is primarily for use in E-mail Forms and 3" receipt printer Forms where the direct-to-port functionality is used. Forms with this format must position the Form Elements using character positions (line and character column), and can't use Regions. Generally you want to define the lines of these Forms in order, top to bottom (the same order it's going to be printed), but this isn't strictly required since the Form's output character array is "built" internally before sending it to the printer. Note that a Character format Form must select a printer type of either Text Output or 3 Printer - Direct to Port.

Character columns

This is only shown for Character format Forms. An appropriate number of columns should be entered so that the "bounds" of the printer is known. This is important for aligning text (e.g. right-justified or centered), and also for auto-wrapping long text.

For 3" receipts, 40 columns us recommended (though some might only work wit 39 columns).

For E-mail, we suggest using 65 columns, though it may work also to put 9999 columns so that text is not wrapped around automatically (assuming the receiver's E-mail program will wrap appropriately) -- and this also assumes that you never use right-justified or centered text, or a Transaction Table.

Trans rows/page

This is only an option for Add-on Forms where a canned receipt is selected. Since these have transaction tables in them (but not as editable elements), you must choose how many rows of transactions can be printed on each page. If the number is too high, transactions may be cut off or overlap text at the bottom of the page. Since some receipts may have a lot of "extra" stuff pushing down the transactions or appearing below it (like credit card info, "Additional sites", or multiple Receipt #'s), it's best to enter a lower number and let it print multiple pages even when there's some extra room.

Printer

This simply determines which printer from the Printer Setup selection is used for printing the Form. If you print everything on a single printer then this may not matter, but it does allow selecting different printers or have different settings for each type of Form. For instance you might prefer to print some Forms in landscape mode and others in portrait mode, so you need to use different Printer selections. Also, Envelopes and of course 3" receipt formats should use the appropriate type.

There are also 4 custom printer selections in case you have special Forms that need specific printer settings.

One of the Printer settings is "Text output (or E-mail)". This must be selected for any E-mail Forms, or any Character-format Forms that don't go to the 3" receipt printer. It can't be used for Variable-format Forms.

Sections (unique pages in a multi-page Form)

The Sections functionality allows for Forms to have multiple pages with different information on each page. Or instance you can create a custom Form with several parts on different pages, and they will all print at once when the Form is printed. Of course most Forms will just have one section.

Note that this does not affect multiple-page printing due to transactions table overflow -- e.g. if it has 2 sections but one section requires 3 pages due to the number of transactions, then it will print that section 3 times (with appropriate transactions) plus the second section, for a total of 4 pages. Of course it also does not affect printing multiple records -- printing a 2-section Form for 3 records will result in 6 pages.

To use the Sections functionality, you need to include the context function **ThisFormSection( )** in the Condition Expressions of your Form elements. Ideally, the main Form will simply have one Region for each section (for logically sub-dividing the Form and making the definition of each section easier), and the Condition Expression for each of those will just check for the section. This each Region is only printed for the appropriate section (page) of the Form -- in other words, the Section is the page number of the Form, not counting the effects of transactions tables overflowing.

For instance: **ThisFormSection( ) = 1** for the first section, and **ThisFormSection( ) = 2** for the second section. If you include this in the region's condition, then there is no need to include it in every element contained in each region.

If the Form is an Add-on Form, the receipt type selected will only be in section 1. So you can add an additional custom section to a canned receipt format, but you cannot create a multi-section Form containing more than one canned format.

Access Level

This is used to restrict Forms to certain operator access levels, for instance if you don't want store clerks to be able to print Window Tags. Just select the minimum access level you want to be able to print the Form.

Base Table

The base table determines the primary data table of the Form -- that is, what type of record this Form is going to show information about. This is mostly used for two things -- where the Form is displayed as an option. and what context functions are available for expressions. Any table in the database may be selected, but only a couple are used for most situations -- Reservations and Customers. If you use the Point of Sale, you might also have Forms for Vendors and Inventory Items.

When printing reservation receipts, for instance, only Forms with "Reservations" as the base table can be displayed (selections are also limited by the Form Type as described above). When using Print from Customer Details, the Form Selection dialog will only show "Customers" based Forms, and so forth.

Regarding the context functions, these follow the general rules for expressions. For instance if "Customers" is the base table, then only **ThisCust** is useful. If "Reservations" is the base table, then you can use **ThisResv**, **ThisCust**, **ThisSite**, and **ThisPark**, because the Form is assumed to be in the context of a reservation and all of that information is known for a reservation. For some tables, such as Inventory Items, only **ThisRecord** is useful (the generic record context function).

E-mail Subject Expression

If the Form Type is "E-mail", then this option will be available where you can enter an expression to be used in the Subject line of the E-mail. If this is blank, then the subject specified in the SMTP setup will be used. Otherwise the expression is executed for the applicable context and the result (assumed to the text) will be used for the subject. If you simply want to specify text, then enter the quoted text as the expression, like this: **"Thank you for visiting"**. However you can also include reservation-specific information using the expression, like: **"Thank you for visiting, " + Cust:Cust_First_Names**.

Save & Test Form

This function does a test Print of the Form, showing a Print Preview window first. This can be used to test the Form repeatedly without completely exiting the Edit function.

You should be able to do most testing just in the preview without actually printing (Close the preview to avoid printing the Form), though some details will require printing to see the actual result. For instance if text seems to be too long or gets cut off, or lines in a table seem to be obscured by text, this could just be an inaccuracy in the preview.

Be aware that it ***does*** completely save any changes you've made to the Form as soon as you click the button, so it negates any possibility of cancelling changes you've made.

Records (for testing)

There's a Records setting next to the Save & Test Form button, which determines how many records (of the given Base Table) are used for testing. Thus if it's set for 10 records, it will attempt to print the Form for 10 reservations, or Customers, or whatever the base table type is. The default is always 10 records, so if you need more or less, be sure to change it before testing. This number will be remembered for each Form.

This is especially useful for Label Forms, but also handy for other Forms. For full-page Forms, this means that multiple pages will be printed -- in the Preview window, use the "Next page" button to step through them. This is a handy way to check Forms for several situations at once, e.g. different types of reservations with different numbers of transactions. But be careful if you actually Print -- it may print more pages than you realize!

Note that it always uses the most recent records for testing, starting from the last record. For instance it will test using the last 10 reservations entered (assuming 10 is entered for records, and "Reservations" base table). This allows you to add some test reservations before testing and it will use those. If you need to test with a specific record, you can go to the editing function for that record (e.g. Customer Details), and print from there using the normal functionality (be sure to enable "Always preview before printing" in Printing Options / Receipts so that you get a preview).

As E-mail (for testing)

For any Forms with "E-mail" as the Form type, this option will appear next to the Save & Test Form button. When this is checked, the Form will be tested in an E-mail window instead of a Print Preview window. This shows it in the proper context, but this also limits testing to a single record. Note that it will be tested with the most recent Reservation for which the customer has an E-mail address (and thus the testing assumed that the Form is reservation-based).

Note that E-mail Forms tested in the Print Preview window will be limited to one page -- anything more than that is cut off. This is because an E-mail Form (or any Character-format Form) is assumed to be unlimited length so it cannot split it up into multiple pages.

Notes

You can enter any notes for yourself here, or use this as a description of the Form -- they're only seen here and in the list in Forms Setup.

Form Elements List

A list of Form Elements appears on the Edit Form Definition dialog, as well as on the Edit Form Element dialog if the element is a Region. It operates the same way in either place, since a Region element is essentially a Form-within-a-Form.

Form elements are actually "Element" records linked to the Form (or region element), so this list shows those linked element records. As with most places where record lists are manipulated, the typical functions are available to Add, Insert, Edit, Copy, Delete, and Move elements in the list. In addition, there are a few special functions as described below.

The element list contains the Condition and Text of the elements, but depending on the element there's no guarantee that either one of these will be filled in. So putting in good Notes for the elements can help later. Also note that the Text column may show different things, e.g. the Expression for data elements or the file name for Bitmap elements.

Note that the Condition and the Text columns may be truncated (with "..." at the end). This is done automatically to limit the column widths and keep long text from making the other columns hard to find. (Even if the columns or the whole dialog is enlarged, it won't show these fields any longer here.)

Form Element Order

When a Form is processed, the elements are processed one after another in the order they appear in this list. Any Region elements are "recursively" processed -- that is, all of a region's sub-elements are processed before moving on to the next element at the same level (more is explained about "levels" below). While this in itself doesn't always make a difference, the order of elements in a Form is actually very important for at least two reasons.

First of all, it's possible for elements to overlap. Think of an element as "painting" on the page. If one element overlaps another, it's painted on top if it and can potentially cover up the previous element. (Regions can partially prevent this for text and data elements, since they clip the text and act like a "cage" for anything written in the region.)

Secondly and possibly more importantly, it's very common to use relative positioning of elements instead of absolute page positioning. For instance, one element can "follow" another so it's placed on the page relative to where the previous element started or stopped. This way you don't always have to figure out exact page positions -- just arrange the elements in the order you want them to appear on the page. Obviously if the order of these is changed, then it will affect their eventual position on the page (or worse, result in an error due to an unknown "previous" position).

Expand Regions

As mentioned above, Region elements are like a Form within a Form. As such, a region has a list of elements of its own that are relative to that region. This can include another region, and so forth, resulting in multiple "levels" of regions, each with their own list of elements.

By default, the Edit Form dialog or Edit Element dialog for a region will only show the elements linked directly to it -- i.e. the current "level" of elements. As you may see in some of the examples, it's common to define an entire Form inside a region (so the margins can be changes easily), and then have various areas of the

Form defined as regions inside that.

The is good organizational practice, but makes it a little more difficult to "explore" the regions to find a particular data or text element -- the top level (the Edit Form dialog) may have just one Region element, then you would need to Edit that region element to see the elements inside it. which may just be a few more sub-regions, and so forth. It may take 3 or 4 levels of "Editing" to get to the actual data and text elements of the Form.

However if you check the Expand Regions box, all regions at or below the current level are expanded into a tree-like list. This allows you to see all elements in the entire Form (or current region) in a single list. To make it easier to read, each level will have a "--->" prefix and will be indented according to the depth, like a tree view. Also, each level will be shaded differently.

You'll notice that some list functions are not available in expanded mode. You can't Insert or Move elements, because it can't handle the multiple-level complexities of this. However you can Copy elements from any level (the new copies will always be placed in the current level). You can also Edit an element from any level, which is the primary reason for expanding the list -- you can quickly locate the element of interest and edit it directly without going through each region above that. (Don't forget that double-clicking an element in the list also does the same thing as Edit).

Import and Export Elements

These functions will import or export any selected elements to a CSV (text) file. The main purpose of this is to copy the elements to a different Form or to a different level (region) of the Form. You can also export commonly used sets of elements for importing to Forms you create later. Basically it's like doing a copy/paste of elements, but going through a file instead of just the clipboard. Note that it's not the same as the import/export function for a complete Form -- it's strictly for copying or moving selected elements. However if an element is a Region, then all of the elements contained in that region are also exported (recursively to any level), and of course they would be imported intact the same way.

When exporting, any elements visible in the list can be exported (even if Expanded as described above), but they must all be of the same expansion level. You cannot select some elements in one level and some elements in another level to be exported at the same time.

When importing, all elements imported are placed in the current level (regardless of expansion). For instance if you're editing a region, then all imported elements are added to that region. If you're editing the Form (the top level), they are added to the Form. Careful manipulation of importing and exporting elements allows you to move elements around between levels, for instance to move some elements to a higher or lower region level.

Editing Form Elements

The Edit Form Element dialog is shown when adding or editing elements from the Edit Form Definition function. It's also used for adding or editing sub-elements for a Region element's Edit Form Element dialog, so you could potentially be multiple levels deep into the same dialog.

Here you edit all of the components of a Form Element. This one dialog is used for all types of elements, even though different information is needed for each type. To help avoid confusion, it will only show the fields that apply to the element type that's currently chosen. Choosing a different Element Type will completely change most of the fields available on the dialog.

The common fields available to all elements are described first, followed by the type-specific fields for each element.

Element Type

Select the type of element you want to show. There are four basic element groups:

Region -- Essentially a Form within a Form, a region contains other elements that are positioned within the region, relative to the region's position. Thus an entire area of a Form can be moved by simply moving the region. Any text elements within the region are also "clipped", so they can't go outside the region. We recommend using regions for any logically similar "part" of a Form to make rearrangement easy. Regions can also be used for the purpose of minimizing condition checks (the region's condition expression only has to be checked once for the whole region, rather than each element), or to help position things in columns (use a region for each column, and text within the region can be auto-wrapped and still stay within the column).

Text / Data -- Anything shown as text characters, either static text or text generated from a Data Expression.

Bitmap / Line / Box graphic elements -- Use a bitmap to include pictures or non-text elements of any kind on the Form. Lines and boxes can also be drawn as needed.

Transaction table -- These are line regions but with specific content, which is basically a canned transaction table. The table can be in several formats. While the flexibility of the data shown in this element is somewhat limited, it's the only easy way of showing transactions on a receipt Form.

Note that Forms with a Format of "Character positions" cannot have regions or graphic elements, and are more limited in the transaction table formats available.

Top / Left

These specify the position of the element on the page (or within the region). Each of these can be specified in one of 5 different ways, which you select from a list. Most will also need a numeric parameter for the position/offset in inches, millimeters or characters depending on the Form's Format, or a percentage value. If it's a character-position Form, character positions start at 1 (e.g. line 1, column 1). For variable-position Forms, positions start at 0 inches or millimeters.

Absolute (in region) -- The numeric value indicates an absolute position on the page (or relative to the parent region's position).

Relative to prev. element -- The value indicates an offset from the position (top / left) of the element before it.

Percentage (of region) -- The numeric value indicates a percentage (e.g. 0 to 100) of the page or parent region's width / height.

Next avail. (+ offset) -- The numeric value indicates an offset from the "ending" position of the previous element. See notes below.

Offset from right/bottom -- The numeric value indicates an offset from the bottom or right limits of the page or parent region.

These allow a lot of flexibility in positioning elements, often without having to know the exact size of a page.

Note that the offset values for "Relative" and "Next avail." positions can also be negative if desired (e.g. to partially overlap or show a superscript, for instance).

"Next avail." positioning

The "Next avail" positioning option is a very important one for text (but also useful for other elements).

For Text and Data elements, the next available position is pretty straightforward (unless the Angle is non-zero as described below). It's the next character or line position, whether the Form is character based or not. For instance when used for a Top position in an element after a text element, it means the next normal line available or single-spaced text. You don't have to worry about the size of the text or anything about where the previous text ended. If you want a little gap, just include an offset of .05 for instance. Or use -.05 to scrunch the text a little closer together. Likewise, using if for the Left position will start at the next available character position. Of course you wouldn't normally use it for both the Top and the Left, just one or the other depending on whether you want the next text to be on the next line below or on the same line right after the previous text.

For other elements, there are some special considerations:

Transaction Tables -- The next available Left position is the right side of the table's area (not necessarily the last character printed), but the next available Top position is the actual next text line position. This is because it's common to want the next text to start just below where the transactions ended, but there's no way to know how many lines will be in the table beforehand.

Lines and Boxes -- The next available top/left position will be the exact bottom/right of the line or box (mostly useful for positioning the next line or box to be drawn).

Bitmap images -- The next available top/left position will be the exact bottom/right of the image according to the element definition. However this is not necessarily the bottom/right of the image as drawn because of the special cases for zero hight or width (see below). In other words if you specify a 0 width, then the next left position will also assume that the bitmap was 0 width even though that really meant to keep the correct aspect ratio. Also note that this may change in the future, so it's best not to use next available positioning after a bitmap at all unless you're specifying the exact size.

Regions -- At the "receiving" end, regions are considered a Form-within-a-Form so the "Next available" position coming into a region, that is for the first element inside the region, will always be zero (which of course is the starting position of the region, not the actual corner of the page). However coming out of a region it remembers the "Next available" position of the last element inside the region. Therefore the next element after the region will know where the actual printing inside the region left off. Thus it's perfectly fine to make a region larger than you expect it might need, and then after the region you can pick up where the last text inside the region actually stopped.

Finally, don't forget the effects of the Condition -- as mentioned below, the "Next avail." position only counts the last element that's actually printed.

Condition

This Expression determines whether the element is included in the Form, and must return a True or False boolean value. If used for a region, it affects all sub-elements in the region also (the entire region is skipped if the condition is False). To edit the condition expression, click on the text box or click the Edit button next to it. The Expression Creator dialog will be used to enter the expression.

Note that if the element is excluded due to the condition, then it's considered non-existent for purposes of positioning also. That is, the "Relative" and "Next Avail." positioning mentioned above refers to the last element that's actually used, not necessarily the one just above it in the list.

Notes

These notes are for your reference, and will also appear in the list of elements. We recommend putting notes in especially for Region elements so you can see what's included in the region, but it's a good idea for any element that's not obvious so you can tell what's going on if you need to edit the Form later.

Region Element Fields

Fields specific to Region elements:

Border

When this is checked, a simple black line will be drawn around the region. Even if you don't plan on having borders in the final Form, this is very helpful in determining whether your region is positioned and sized properly while you're testing it.

Bottom / Right

Regions must have a bottom and right boundary. As with the Top / Left position there are several ways to specify the bottom / right boundaries, with a few minor differences as described here:

Relative (height / width) -- The value indicates an offset from the Top / Left position of **this** element (so it's effectively a height or width value).

Absolute (in region) -- The numeric value indicates an absolute position on the page (or relative to the parent region's position).

Percentage (of region) -- The numeric value indicates a percentage (e.g. 0 to 100) of the page or parent region's width / height.

Percentage, relative -- The numeric value indicates a percentage of the space remaining on the page or parent region, starting from the top / left of the region.

Offset from right/bottom -- The numeric value indicates an offset from the bottom or right limits of the page or parent region.

Sub-elements in Region

All elements contained within the region are shown here, with typical editing functions. This works the same way as the top-level Form Elements List, so refer to that section for details.

Text and Data Expression Element Fields

These fields are specific to text and data elements (with only a couple differences as mentioned below).

If the Form uses Character-position format, most of the font and formatting options are not available -- only the alignment and auto-wrap options can be used. Also note that for Forms using direct-to-port printing, the Text (or Expression result) may have "escaped" hexadecimal characters such as for auto-cutter or cash drawer control. These must be in the form \xHH, for instance **\x07** to send the BEL character. Note the direction of the slash (backslash), and that the "x" is lower-case. Also note that if any such escape

sequences appear in center-aligned text, the alignment is based on the resulting single character, not the 4-character escape sequence.

Text (only in Text elements)

This is the static text to be displayed. It can actually be any length -- click the Edit Text button to open a larger window to edit long text. Note that the text can also include multiple lines, as long as the Auto-wrap option is also checked (of course this also results in auto-wrapping any lines too long for the region or page). Multiple lines will be shown as-is in the large editing window, but the line breaks will be shown as " \\ " in the single-line edit box.

Expression (only in Data Expression elements)

This is an expression which will be executed and the results will be shown as text, subject to the Format below. Once the expression is evaluated and formatted, all other aspects of the element work the same as a text element (e.g. the return value of the expression becomes the "static text" to be shown). To edit the Expression, click on the expression text box or click the Edit Expression button. The Expression Creator dialog will be used to edit the expression.

Format (only in Data Expression elements)

This determines the text formatting of numeric values, assuming the expression results in a number. For non-numeric data, the "General" option should be chosen. For numeric data, select an appropriate format, e.g. Currency, Integer, Percent, or Floating Point. If none of these quite fits your needs, then you can select Custom Format and enter an expression to format the data any way you need to.

Custom Format Expression (only in Data Expression elements)

This field only appears when the Format selected above is "Custom Format". To edit the format, click the Edit Custom Format.. button or simply click on the text box below it. This invokes the Expression Creator dialog to edit the format expression.

The format expression should simply convert a numeric value to text. The numeric value to be formatted will be available with the context function **ThisValue**, which will already be shown in the expression the first time you edit it. There are several functions available for formatting numeric values. For a simple example, lets say that we want to take the number an show it as a currency value rounded to the nearest dollar. The expression entered would be:

```
Currency( Round( ThisValue(), 0))
```

Color

This determines the color of the text and the background of the text. An example is shown with the current colors selected. Click the Text or Background button to change the combination as needed, and these will allow you to choose any color. Of course black text and white background is the default, and the background is typically white (unless you want to use a lot of ink!).

Font

Any font that's installed in Windows can be used, in any size supported, with any combination of Bold, Italic, or Underlined attributes. The most common font names are "Arial" and "Times new Roman", and occasionally "Courier New" for fixed-pitch text. To change the font using the standard Windows font-selection dialog (which usually shows an example of the font), click the **Select Font** button.

Note that new elements will default to the most recent font selected, or the last font used in a Form element, so you don't have to constantly change the font for every element in the Form.

Shrink to fit available width

When this is selected, it can automatically resize the text to keep it on a single line, within the borders of the page or parent region. You also specify the smallest font size to which you'll allow it to shrink (anything smaller than 8 can be very hard to read). If even the smallest size doesn't allow it to fit, then the text will be truncated (this can't be used with the auto-wrap option).

Align Text

Choose how to align the text on the page (or within the parent region), either Left, Right, or Center. You can also choose to center the text vertically within the region.

Auto-wrap

When this is selected, text that's too long for one line will auto-wrap (without breaking words) to multiple lines as needed. This is also required for supporting text with forced line breaks in it -- essentially any text that you don't want limited to a single line.

Angle

Using this is rather tricky, but it does allow support for drawing text at any angle. Enter a number from 0 to 360 (0 is normal horizontal, 90 is "up", 180 is upside down, etc.). Note that the text may or may not obey the bounds of any region, and the Auto-wrap, Alignment, and Shrink-to-fit options will be ignored. Only single lines are supported. Also, the "Next Avail" positioning of the next element may or may not have the results expected.

Basically it's fine for single-line text, but if you want multi-line text you'll need to manually figure out the absolute position for each line for the given angle, and how much will fit on each line.

Bitmap Element Fields

The following fields are specific to bitmaps:

File name

Enter the file name or use Browse to select it. Note however that the file must reside in the same folder as the current database, and it's not backed up with the database (basically just like Map files). Thus you may need to copy any files used to all computers using Campground Master.

Bottom / Right

Bitmaps must have a bottom and right boundary. This indicates the area on the Form that the image will occupy, and if the original image is not exactly that size then it will be stretched to fit in this space (height and width are stretched separately, which can distort the image).

The same Bottom / Right options are available here as for Region elements, so refer to those above for details.

However there are a couple special case for bitmaps -- if both a height and width of 0 is specified (e.g. using the Relative option), then the actual image height and width is used for that dimension. This allows a 1:1 image sizing, however this will be in "pixels" -- so it may appear smaller than expected on the page.

Another option is to use one of the other options for one dimension and a Relative / 0 option for the other. The image will be stretched to fit the non-zero height or width specified, and also stretched in the other dimension to keep the aspect ratio. For an example, if you want the image to be in the upper left corner and exactly 4 inches wide, keeping the correct aspect ratio, then use the following settings:

Top : Absolute / 0
Left : Absolute / 0
Bottom : Relative / 0 (which means keep the aspect ratio)
Right : Absolute / 4.0 (make it 4 inches wide)

Line Element Fields

The following fields are specific to lines:

Bottom / Right

Lines must have a bottom and right position to specify the "end" of the line. The line will be drawn straight from the top/left to the bottom/right, which can be any direction. The same Bottom / Right options are available here as for Region elements, so refer to those above for details.

Color

This determines the color of the line. An example is shown with the current colors selected. Click the Line button to change the color of the line. Note that the (background) is not actually used in drawing the line, but if the line is to be white then you can select a black "background" just so the example shows up.

Line width

Enter a number for the width of the line. Of course 1 is the skinniest. The number is generally in printer-dots, so the actual width in physical size may depend on the printer used.

Box Element Fields

The following fields are specific to boxes:

Bottom / Right

Boxes must have a bottom and right position to specify the other corner of the box. The same Bottom / Right options are available here as for Region elements, so refer to those above for details.

Color

This determines the color of the outline of the box and the fill color. An example is shown with the current colors selected. Click the Outline button to change the color of the line around the box, and the Fill button to change the interior filled color of the box.

Line width

Enter a number for the width of the outline of the box.

Transaction Table Element Fields

The following fields are specific to transaction table elements. Note that for character-position format Forms, only the Bottom/Right and Table Type fields are available and the narrow paper format option is assumed.

Bottom / Right

Transaction tables must have a bottom and right position, similar to regions, which specifies the bounds of the table . The same Bottom / Right options are available here as for Region elements, so refer to those above for details.

Font / Shrink to fit

You can choose the font as well as the shrink-to-fit option just as for text elements above. Note however that the shrink-to-fit option applies to each cell of the table individually, not the table as a whole. Also, the font attributes for bold, italic and underlined cannot be chosen for table.

Table Type

There are four different types of tables available (only two of them can be used for character format Forms).

Standard receipt -- The basic receipt-type table used for most receipts, invoices, etc., where the charges and discounts are shown on the first section, then taxes, then any credits, and finally any payments.

Register-style statement -- In this format, the transactions are shown in their original order, with columns for Charges, Payments, and Balance after each transaction.

Credit card slip -- This is designed to just show the credit card payment -- it will only show the mos recent payment transaction on the slip with the total amount paid.

Purchase order -- This is for use only in Point of Sale purchase orders, where it uses purchase order transactions instead of customer transactions.

Narrow paper format

This flag is generally for 3" receipt printer output where the space is limited, but is also used for E-mail where a variable positioning of columns is not possible. It uses two lines for transactions, with the transaction description on one line and the quantity, each and total on a second line. The total will be right-justified according to the number of columns (or boundary of the table element).

Grid lines

Select this option if you want grid lines for the table. Don't forget that this can make receipts print much slower on ink jet printers.

Shade / color cells

Select this option if you want the Total column background shaded grey and the Total / Balance label in a black background. Don't forget that this does take more ink.

Shrink to fit more lines

This option allows it to use an overall smaller font for the table as the number of lines increases, so more lines can fit on a page. It's more efficient than printing multiple pages while allowing short receipts to have larger text, but it does make them less consistent. When this is selected, you also specify the minimum font size to use. It will shrink the font when needed to fit the rows within the table's area.

Max rows / page

You need to specify the maximum number of that can be printed rows per page. If the number is too high, transactions may be cut off or overlap text at the bottom of the page. Since some receipts may have a lot of "extra" stuff pushing down the transactions or appearing below it (like credit card info, "Additional sites", or multiple Receipt #'s), it's best to enter a lower number and let it print multiple pages even when there's some extra room.

Note that if the Shrink-to-fit option is enabled, assume that the smaller text size will be used by the time it reaches the maximum, and do some testing to see how many you can really fit with the maximum amount of "other" stuff printed (resulting in the smallest area available for the table).

Date / Qty / Each / Total widths

Enter the width desired for each of these columns. You may enter 0 for the Date width to exclude that column, but the other columns will always be included. You could enter 0 for their widths, but some remnants would still remain.

Note that for the Register-style statement, there is no Qty or Each column so this information is added to the Description, like "(6 @ $10.00)". The Each width is used for both the Charges and Payments columns, and the Total width is used for the Balance column.

You'll notice that there's no setting for the Description column -- the Description column will use any of the table's overall width left over after sizing the other columns.

Form Selection Dialog

A general Form Selection dialog is used in many places throughout Campground Master, typically for selecting a Form to print (e.g. from Find Reservations, Customer Details, Sites Setup, etc) or for selecting an E-mail format to use (from Transactions / E-mail Confirmation).

In some cases, the Form Selection dialog will also have an option at the top to print a Grid format instead of a Form. This will typically print a list of fields for the current record (e.g. if invoked from a single-record dialog like Edit Site or Customer Details), or it will print the entire list of records being shown (e.g. if invoked from a multiple-record dialog like Find Customers).

If the grid option is not selected (or not available), then a drop-down list will contain one or more Form types to choose from. When a Form type is selected, any available Forms of that type will be shown on the list below. Just double-click on that Form (or select it and click the OK, Print, or Use button as appropriate) to use that Form.

There may also be an Edit Form button available if the current operator has sufficient access permissions to edit Forms. This allows you to edit the Form and then come back to print it again, without going back to the Maintenance functions.

Note that if Print Preview is enabled for receipts (through Printing Options), then of course it will show a preview of the Form when the Print button is clicked. An added benefit is that if you Cancel the preview instead of Printing, you'll still be in the Form Selection dialog where you can choose another Form, or Edit the Form and try again.

Menus

Overview

Almost any of the drop-down or right-click menus in Campground Master can be customized with additional functions defined by expressions. Any number of menu selections can be added to the menus, which can also be organized in sub-menus any number of levels deep.

In addition, existing selections in the menus can be conditionally renamed or deleted to clarify or simplify the functions seen by most users.

Note that all menu customizations are done within the existing menu structure, modifying the standard menus -- it's not possible to create a completely "new" menu, simply because the program wouldn't know when or where to show it. However you can modify the very top menu bar, which means you can add a new drop-down menu to it (as well as add sub-menus in any of the drop-down menus), and you can modify almost any of the right-click menus in the system to add functions or sub-menus.

Menus Setup

To modify a Menu, go to Maintenance / Advanced Customizations / Menus. This opens the Menus Setup dialog, which lists all current Menu Definitions and has the typical functions for Adding, Inserting, Editing, etc. Note that while there are functions to Move Up and Move Down, a Menu Definition's position in the list does not usually affect the functionality. However if you create multiple definitions which modify the same menu, then the order in the list can determine the position of any new Menu Items added to the menu.

You can also Export one or more Menu Definitions to a text file, or Import Menu Definitions. This is primarily used for importing Menu Definitions created by the software provider, though it can also be used to transfer them between multiple databases.

> **Note:** We refer to the entries as "Menu Definitions", but they're not actually "new" menus and don't even necessarily add anything to a menu (it could just modify the text of an existing selection in a menu). A Menu Definition is simply a list of "Items" to change in a specific menu, where each "Item" defines some change to the selected menu. The change could be to add a new selection in the menu which performs an "Action" when it's clicked, or the change might just be a command to remove or rename an existing selection, or it could be a complete sub-menu structure.

Menu Definitions cannot have duplicate names (this is primarily just to avoid confusion during setup). If you make a Copy of a Menu Definition, text like "(copy 1)" will be added to the name to make sure it's unique. Of course you can change this to be more appropriate. Duplicate checking for the names is not case-sensitive ("My Menu" is considered the same as "my menu").

Some common modifications are described below, to give you an idea of what can be done.

Removing menu selections (e.g. based on access levels)

While most functions in the menus can have their allowable access level changed through the standard Access Levels setup (Maintenance / Park Setup / Access Levels), this will only disable the menu selection rather than remove it from the menu. If you prefer to remove the disallowed functions, in order to simplify the menus for your users, then you can do it through the Menu Definitions.

To disable a menu selection, Add a Menu Definition for the appropriate Base menu. Then in the Edit Menu dialog, Add a Menu Item. In the Edit Menu Item dialog, select "Remove selection". Now select the option to find the selection by "ID/Command", and choose the menu selection form the large drop-down list.

Now set the Condition expression for when to remove the item. (If the Condition is blank, it will always remove it.) For instance if you only want to remove the selection if the current operator has a low access level (e.g. non-administrator), then you would enter a Condition expression like this (note that access levels are 0 to 5, where 0 = Guest and 5 is Administrator):

```
FieldValue(CurrentOpRec(),"Oper_Access") > 4
```

You can add as many Menu Items as needed in the same Menu Definition, to remove other items in the same menu.

Renaming menu items

If you prefer to use different text in the menus, either for clarification or to localize it to your language as much as possible, you can use the same procedure as described above for removing items. The only difference is that you would select "Rename selection" instead of "Remove selection" as the item type, and you would probably not have a Condition. Then just enter the new name in the Selection Name field.

Adding new functions to a menu

While the possible actions are limited by what can be defined with Expressions, there are many expression functions available for typical operations that will probably do anything you would want to do.

For instance there may be something you do often that takes several steps, which could be simplified by adding a custom Action to a menu which does it all at once. Or you might need to do some complex manipulations with the data on a regular basis and output that to a file -- this could be done with a Script, and adding a Menu Item somewhere to execute the Script makes it easy to perform the function whenever you need to. Likewise, you could read a file and process the data in a Script, such as reading the output of a phone system and adding the charges to the appropriate reservations.

Remember that right-click menus will have the context available for the expression defining the action, so for instance you can access the specific reservation that was clicked on. Top-level menus don't have any context to work with, but potentially some very complex actions could be created through scripts -- for instance to scan the entire database for a certain condition and show that information in a pop-up window or even a custom Dialog.

As a quick example, lets say that you want to add a command to the right-click menus to send a simple "Thank you for staying" E-mail (e.g. you could do this whenever you check them out).

First Add a Menu Definition for the appropriate Base menu -- in this case, "Right-click - Reservation, on other views" so it appears in the Departures tab view (you could do the same for the Rack right-click menu). Then in the Edit Menu dialog, Add a Menu Item. In the Edit Menu Item dialog, select "Execute Action". Enter an appropriate Selection Name, like "Send Thank-you E-mail".

Naturally this only makes sense if the customer has an E-mail address, and lets assume we also don't want this selection to appear in the menu until after the Check-out has been done. We can add this Condition so that the selection only appears when needed:

```
Cust:Cust_Email != "" AND Resv:Resv_Status = "Checked Out"
```

Now add the Action Expression, which will use the SendEmail function. (Note that this only works if you have the SMTP E-mail settings configured -- see Online Setup.) Obviously a real message would be longer than the one below, and you may want to include some fields from the reservation in it to personalize the message, but this should give you the general idea:

```
SendEmail(.T., "", "", Cust:Cust_First_Names, Cust:Cust_Email,
"Re: Your recent stay", "Thank you for staying with us.  Come Again!")
```

Save that Menu Item and you'll be back in the Edit Menu Definition dialog. Here you might want to fill in the "Insert at position" field so that the item you just added doesn't go at the very bottom. For instance, a value of 8 should put it right after the Check Out function on the Departures tab. Then Save the menu definition.

Editing Menus

The Edit Menu Definition dialog is shown when adding or editing Menus from Menus Setup.

Here you can view and edit all of the Items in a Menu Definition. There are a few fields you edit directly here, and the rest of the dialog lists the Menu Items.

Menu Definition Name

The name should be descriptive enough for identifying the Menu in Menus Setup, but that's generally the only place it's used. Each Menu Definition must have a unique name (which is not case-sensitive).

Enabled

The Menu Definition can be disabled by unchecking this box, so that it does not get processed. Note that if it's disabled, it doesn't just mean that the items added to the menu are disabled (greyed out) -- it means that the Menu Items in the definition won't be processed. Therefore it won't modify the base menu at all, as if the definition is not even there.

Modify Base Menu

Each Menu Definition is simply a list of things to do to an existing menu, so you need to select which menu it will modify from this list.

Main menu definitions can modify the top level menu (the menu bar across the top) or any one of the drop-down menus from there. Note that only the direct (first-level) drop-down menus can be selected, so any additions to the menu can only be made to that level. Of course you can add your own sub-menu at that level, and the sub-menu can contain any number of levels beneath it. However you can't add an item directly inside the Printer Setup under the File menu, for instance.

Note that while you can't add new selections (e.g. Actions or Sub-menus) at lower levels, it's still possible to modify existing selections at any level in the menus. For Remove or Rename item types you can choose the menu selection (ID) at any level below the base menu. So for instance you could completely remove all Printer Setup selections for non-administrator users.

Most right-click menus can also be selected as the Base Menu to be modified. Some right-click menus are specific to a particular view (e.g. for the Rack or Map) because the conditions there are unique, while the other views and situations are more general -- those base menu selections will affect the menu on any of the other tab views. For instance the same "Transactions" right-click menu is used for any view where a Transaction is selected, e.g. in the Transaction tab view and any user-defined Query based on Transactions.

Context for Expressions

Any right-click menu will have some context information available for the expressions in its Menu Items. Naturally the information available depends on which menu is the selected as the base, and in some cases the record selected when right-clicking will determine the context available.

For instance in a Transactions menu, **ThisTran( )** will always be available, plus there might be **ThisCust( )** and **ThisResv( )** context info if the transaction is for a customer or reservation. A Customer menu will only have **ThisCust( )** available, and the right-click for queries of "Other record types" will just have the generic **ThisRecord( )**.

On the Rack, **ThisDate( )** will be available for the cell clicked (and **ThisPeriod( )**, if applicable, when viewing Scheduled reservations).

The Cross table Query menu will have a list of records "included" in the clicked cell (e.g. used for the calculation in that cell), so **ThisRecList( )** and **ThisRecCount( )** are available to access those.

The context functions **ThisFromDate( )** and **ThisToDate( )** can be used to get the From and To dates selected in the tab view or query report.

Menu Items List

A list of Menu Items appears on the Edit Menu Definition dialog, as well as on the Edit Menu Item dialog if the Menu Item is a Sub-menu. It operates the same way in either place, since a Sub-menu Item is

essentially a Menu-within-a-Menu.

Menu items are actually "Item" records linked to the Menu (or sub-menu item), so this list shows those linked item records. As with most places where record lists are manipulated, the typical functions are available to Add, Insert, Edit, Copy, Delete, and Move items in the list. In addition, there are a few special functions as described below.

The item list shows the Condition and Text of the items, but depending on the item there's no guarantee that either one of these will be filled in. So putting in good Notes for the items is recommended.

Note that the text in some columns may be truncated (with "..." at the end). This is done automatically to limit the column widths and keep long text from making the other columns hard to find. (Even if the columns or the whole dialog is enlarged, it won't show these fields any longer here.)

Menu Item Order

When a Menu Definition is processed, the items are processed one after another in the order they appear in this list. Any Sub-menu items are "recursively" processed -- that is, all of a sub-menu's sub-items are processed before moving on to the next item at the same level (more is explained about "levels" below). While this in itself doesn't always make a difference, the order of items in a Menu Definition determines the order they're added to the menu (assuming the item isn't just a Rename or Remove item).

Expand Sub-menus

As mentioned above, a Sub-menu item is like a Menu within a Menu. As such, a sub-menu has a list of items of its own that are relative to that sub-menu. This can include another sub-menu, and so forth, resulting in multiple "levels" of sub-menus, each with their own list of items.

By default, the Edit Menu dialog or Edit Item dialog for a sub-menu will only show the items linked directly to it -- i.e. the current "level" of items.

However if you check the Expand Sub-menus box, all sub-menus at or below the current level are expanded into a tree-like list. This allows you to see all items in the entire Menu Definition (or current sub-menu) in a single list. To make it easier to read, each level will have a "--->" prefix and will be indented according to the depth, like a tree view. Also, each level will be shaded differently.

You'll notice that some functions are not available in expanded mode. You can't Insert or Move items, because it can't handle the multiple-level complexities of this. However you can Copy items from any level (the new copies will always be placed in the current level). You can also Edit an item from any level, which is the primary reason for expanding the list -- you can quickly locate the item of interest and edit it directly without going through each sub-menu above that. (Don't forget that double-clicking an item in the list also does the same thing as Edit).

Import and Export Items

These functions will import or export any selected items to a CSV (text) file. The main purpose of this is to copy the items to a different Menu Definition or to a different level (sub-menu) of the Menu Definition. You can also export commonly used sets of items, for importing to Menu Definitions you create later. Basically it's like doing a copy/paste of items, but going through a file instead of just the clipboard. Note that it's not the same as the import/export function for a complete Menu Definition -- it's strictly for copying or moving selected items. However if an item is a Sub-menu, then all of the items contained in that sub-menu are also exported (recursively to any level), and of course they would be imported intact the same way.

When exporting, any items visible in the list can be exported (even if Expanded as described above), but

they must all be of the same expansion level. You cannot select some items in one level and some items in another level to be exported at the same time.

When importing, all items imported are placed in the current level (regardless of expansion). For instance if you're editing a sub-menu, then all imported items are added to that sub-menu. If you're editing the Menu Definition (the top level), they are added to the end of the definition. Careful manipulation of importing and exporting items allows you to move items around between levels, for instance to move some items to a higher or lower sub-menu level.

Editing Menu Items

The Edit Menu Item dialog is shown when adding or editing items from the Edit Menu Definition function. It's also used for adding or editing sub-items, from a Sub-menu item's Edit Menu Item dialog, so you could potentially be multiple levels deep into the same dialog.

Here you edit all of the components of a Menu Item. This one dialog is used for all types of items, even though different information is needed for each type. To help avoid confusion, it will only show the fields that apply to the item type that's currently chosen. Choosing a different Menu Item Type will completely change most of the fields available on the dialog.

The common fields available to all items are described first, followed by the type-specific fields for each item type.

Menu Item Type

The item type defines what will be done to the base menu -- either adding something to it (an action, separator or sub-menu) or changing something already in it (remove or rename a selection).

Execute Action -- This adds a selection to the menu, which executes an expression when the selection is clicked on.

Sub-menu -- This adds a selection which doesn't do anything itself, but opens another menu. You also define all of the selections which will be in that sub-menu.

Separator -- This just adds a line in the menu, used for separating groups of selections.

Remove selection -- This will completely remove an existing selection from the menu.

Rename selection -- This will change the name of an existing selection in the menu.

Condition Expression

All Menu Item types can have an optional condition expression. If a condition is entered, this determines whether the item is processed, e.g. whether the selection is added or not, or whether the selection is removed/renamed or not. It doesn't just disable the item (that's done by access level -- see below).

The condition expression must return a True or False boolean value. If used for a sub-menu, it naturally affects all sub-items too (the entire sub-menu is only added if the condition is True). To edit the condition expression, click on the text box or click the Edit button next to it. The Expression Creator dialog will be used to enter the expression.

Notes

You can add Notes to any item to help clarify its use. The notes will only show up in the list of menu items (the user will never see the notes).

Execute Action Items

Action Items are the main part of a menu -- the selections that actually do something useful. Anything you see in a menu that's not a sub-menu or separator is an "Action" item, whether it does something immediately (e.g. Save the database file) or opens a dialog prompting for more information (e.g. Log In).

Access Level

Select the lowest access level that should be able to do this action. If the current operator does not have this level or higher, the selection will still be in the menu but it will be disabled (greyed out).

Menu Item Selection Name

This is the text that will appear for the action item in the menu. You can either enter static (non-changing) text, like "Delete reservation", or you can create an expression such that the menu text will change depending on the situation or the record in question (e.g. "Extend by 1 month (to April 30)").

You need to select which way you want to enter the selection name (click one of the radio buttons), and then the appropriate entry field will be shown. If it's static text, then just enter it in the Selection Name field. If you choose to enter it as an expression, the Selection Name box will be greyed out -- to edit the condition expression, just click on the greyed-out box or click the Edit button next to it. The Expression Creator dialog will be used to enter the expression.

Underlining Hot-key letters

You can specify which letter is underlined, to be used as a hot-key for the menu selection, by inserting an ampersand ("**&**") in the text. For instance if you enter "Delete re&servation", the "s" will be underlined and will be the hot-key, and it will be shown as "Delete reservation". Then in the resulting menu you can just press "s" on the keyboard instead of using the mouse to click on the selection in the menu.

Be careful about underlining a letter that's already used as a hot-key. If there's more than one selection in the same menu that uses the same hot-key letter, pressing that key will only move to that selection (highlight it) -- it won't execute it. Pressing the key again would move to the next one with that hot-key, etc.

Note: If you actually want to show "&" in the menu, like "This & That", then you need to use two of them in the selection text: "This && That". This tells Windows that you want to show the & instead of underline the next letter.

Important: There's an option in Windows XP to hide the underlines until you press the Alt-key, and in Microsoft's wisdom they seem to have enabled that option by default in some cases (why confuse the user with all those underlines, right?). So if you don't see any letters underlined in the menu, just press the Alt key and the underlines should show up (this applies to all menu selections, not just the ones you add). You can also disable this option in XP by going into Desktop Properties (right-click on an empty desktop area), click the "Appearance" tab, click the "Effects" button, and uncheck the "Hide underlined letters" option.

Action Expression

This defines what will actually happen when the menu item is selected. Simply put, this expression is executed when the item is clicked. The result of the expression is not used, so it doesn't have to return any particular type of value. To edit the action expression, click on the text box or click the Edit button next to it. The Expression Creator dialog will be used to enter the expression.

Naturally the expression should "do" something useful. Many useful functions are available to do things like show a message, open Queries, send E-mail, print Forms, open Dialogs, or even just execute another menu command. To browse some of the possible actions, select the "User interaction" function type (through Insert Expression Element, in the Expression Creator dialog).

Of course it doesn't necessarily have to do anything visible -- for instance you can just set a field value. But we recommend that you always show some indication that the action was performed, e.g. with a **MessageBox( )** confirming the action and perhaps showing the details or results, so the operator knows that something actually happened and that they clicked the right thing.

If you do anything complex, e.g. more than one function, then it may be a good idea to create a Script for it and then just use **CallScript( )** to call the Script from the Action expression here.

Sub-Menu Items

A sub-menu is used to organize menu functions in logical groups, and to keep each level of a complex menu from getting too long. A sub-menu will have a name in the menu just like an action item, but clicking on it will simply open another menu (usually to the right of the sub-menu item, if there's room for it there). A right-arrow indicator will be added to the right edge of the menu automatically (by Windows), to indicate that it will open another menu.

A sub-menu item needs a Selection Name to be shown in the menu, which is described above for Execute Action items (see "Menu Item Selection Name" above).

A sub-menu also needs Menu Items of its own, so a list of Menu Items will be shown here with applicable editing commends, just like the Edit Menu Item dialog. See the previous section about the Menu Items List for details.

Separator Items

A separator is just a line in the menu used for organizing selections into groups, so there's no more information needed for a separator.

Remove Selection / Rename Selection Items

These item types are used for modifying existing selections in a menu. Generally this is done for standard selections that the program includes, but it's also possible to do it for selections that you've add yourself (though it's better to handle this with the Condition or Selection Name expression of the original Menu Item definition).

If it's a Rename Selection item, it needs a new Selection Name that the original menu selection will be renamed to. This field is described above for Execute Action items (see "Menu Item Selection Name" above).

Finding the Item to remove/rename

For both Rename and Remove selection items, it needs to know which item to act on. You can have it find the item by name or by its internal ID/command, so first select which way you'll find it.

Whenever possible, e.g. whenever it's for a standard menu command, choose the ID/Command option and select the appropriate command form the list. The list shows all menus in the system that can be modified, with all sub-menu command items. Note that the "ID to remove" will be shown for the selected item, but only for reference -- it cannot be modified directly.

Tip: You can use this to find out the command ID of a menu item, for instance if you want to execute the command through the expression function **MenuCommand( )**.

While you could choose to find the item by name, that might not always work -- sometimes a selection's name is changed based on the condition (e.g. "Check out" vs. "Undo check-out"), and it's also possible that the menu selection name may change in future versions of the software (but the command ID should never change).

There are some cases where you do need to find it by name. If it's an item you added through a Menu Definition yourself, the ID is not known at this time so the name must be entered. Also, sub-menu names don't have an ID, so they would have to be located by name. For instance if you wanted to rename the "Park Setup" sub-menu selection under Maintenance, you would have to find it by name.

Note that in either case, the "search" for the selection is recursive -- it will look at all levels of sub-menus below the base menu of the Menu Definition until it locates the selection to be renamed or removed. Of course if it doesn't find a match, then nothing happens.

Important: When searching by name, any ampersand characters ("&") in the menu selection will be ignored. For instance if the original menu selection name is "&File" so that it's shown as "File" in the menu, you should only enter "File" in the text to search for.

Dialogs

Overview

Through the Dialogs Setup functionality, several of the main data entry dialogs can be customized to a large extent -- new entry fields can be added to a dialog (e.g. for new data fields added to a table through Data Field Definitions), existing controls can be disabled, deleted, moved or renamed, and the dialog can be enlarged to accommodate new fields. Additional actions can be taken when any control is activated (e.g. when a button is clicked, edit field changed, etc.), supplementing the normal program operation with your own special functionality.

In addition, completely new dialogs can be created for displaying information or for data entry. A custom dialog can be displayed by calling the Expressions function **DlgOpenUserDialog( )**, for instance when a new button is clicked on another dialog, or through a custom Menu Definition or when an Event Action is triggered.

As with all advanced customizations, this requires a rather technical understanding of the system. **It must also be emphasized that modifications to existing Dialogs are more "dangerous" than most other customizations because they change the operation of the program, and can also cause a system malfunction.**

Dialogs Setup

A "Dialog Definition" is a list of instructions for modifying a particular dialog. Create a Dialog Definition for each system dialog that you want to modify, or for each new dialog you want to add.

Note that only one Dialog Definition can be enabled for each system dialog, e.g. for the Edit Reservation dialog, but the definition can contain as many Dialog Elements as needed to make all of the customizations you want.

To create a Dialog Definition, go to Maintenance / Advanced Customizations / Dialogs. This opens the Dialogs Setup dialog, which lists all current Dialog Definitions and has the typical functions for Adding, Editing, etc. Since the order of the definitions doesn't affect any functionality, there is no Insert command -- but you can Move the dialogs up or down in the list if you want to rearrange them.

You can also Export one or more Dialog Definitions to a text file, or Import Dialog Definitions. This is primarily for you to import Dialog Definitions created by the software provider, though it can also be used to transfer Dialog Definitions between multiple databases.

Dialog Definitions cannot have duplicate names. If you make a Copy of a Dialog Definition, text like "(copy 1)" will be added to the name to make sure it's unique. Of course you can change this to be more appropriate. Duplicate checking for the names is not case-sensitive ("My Dialog" is considered the same as "my dialog").

Expression Functions for Dialogs

Most expressions in dialog element definitions need to do something with the controls in the dialog -- such as get and/or set a dialog control's current value, change which control has focus, modify controls or perform other functions within the dialog. A whole set of functions are available for this, all of which start with the letters "**Dlg"**. Refer to the function types "User-defined dialogs" in the Expression Elements dialog (through the Insert Expression Element function of the Expression Creator) for all of the functions available for accessing and working with dialog controls.

Note that there are separate sets of functions for new controls (added by the dialog definition) vs. standard controls (in an existing dialog), because new controls must be referenced by name whereas existing controls must be referenced by a numeric control ID. Be sure to use the functions with "User" in the name to access a new control added in this dialog definition, and use the function without "User" in the name for standard controls that are in the existing dialog.

"Changed" Flag

Each dialog keeps track of a "changed" flag so that it knows whether anything needs to be saved back to the record when it's closed. When you add a new control to a dialog, the program already handles this flag internally for obvious changes, e.g. when the value of the control is changed. However there may be times that you need to change something internally and need to let the dialog know it. There is an expression function **DlgSetChangedFlag( )** for setting the "Changed" flag when needed. There's also a function to get the current status of the flag, in case you want to see if something has changed before performing an action of your own.

Opening a Custom Dialog

Once you've created a new dialog, you need some way to open it. Use the function **DlgOpenUserDialog( )** with the dialog name to open the dialog. You can also give it the record to be used (in context). You might want to do this from a Menu definition, an Event, or from a button action in another dialog.

This function returns True if the "Save" or "OK" button is clicked (or more specifically if the **DlgClickSave( )** function is called within the dialog expressions), or else it returns False if "Cancel" is clicked or if it's closed with the "X" button.

Editing Dialogs

The Edit Dialog Definition dialog is shown when adding or editing Menus from Dialogs Setup.

Here you can view and edit all of the Items in a Menu Definition. There are a few fields you edit directly for the definition, and the rest of the dialog lists the Dialog Elements.

Dialog Name

The name should at least be descriptive enough for identifying the Dialog in Dialogs Setup, but if it's a new dialog rather than an Add-on, then this name will also be the title of the dialog when it's displayed and also the name used to open it from the **DlgOpenUserDialog( )** function. Each Dialog Definition must have a unique name, whether it's an Add-on or not. The name is not case-sensitive, so "My Dialog" is considered the same as "my dialog".

Add-On

To define changes or additions to a standard dialog, check this box and select the appropriate dialog from the list. To create a brand new dialog, leave this box unchecked.

Only one Add-on Dialog Definition can be Enabled for each standard dialog, e.g. for the Reservation Details dialog -- but you can actually create multiple definitions for the same dialog as long as only one is Enabled at a time. Thus you could experiment with a new definition without taking it "live" by changing which one is enabled. (Technically, with some clever expression scripting, you could have multiple definitions that are enabled & disabled dynamically depending on other conditions -- so it's possible to have more dynamic changes, but that's too advanced to be covered in detail here.)

Enabled (Add-on only)

An Add-on Dialog Definition can be disabled by unchecking this box, so that it does not get processed. This is only applicable to Add-on dialogs, because non-add-on dialog definitions aren't automatically used (only opened explicitly through an expression function). If an add-on dialog definition is disabled then it won't modify the base dialog at all, as if the definition is not even there.

Base Table (non-Add-on only)

A Dialog Definition that's not an Add-on needs to have a Base Table selected (any Add-on has a base table already defined by the dialog it's modifying). Select the table for the type of record will be shown (and possibly modified) in the dialog. In general, any new dialog is assumed to be operating on a single record of the selected type, and this context record is specified when the dialog is created through the **DlgOpenUserDialog( )** function.

The Base Table selection also affects which fields are available in the Quick-Add Fields function (described in the Dialog Elements List section to follow).

Width & Height

The units for these values are screen pixels, so you should know the resolution of the screen (or screens) to be used. As a general guide, Campground Master typically uses dialogs that will fit on 640x480 screens, but some will be sized for 800x600 screens automatically if the current resolution is high enough. Since most modern systems have a resolution of 1024x768 or higher, it's possible that you can use much larger dialogs to allow for more fields.

Note: If you're using Campground Master on multiple computers, be sure to design for the **lowest** resolution in use, otherwise the dialog may not be completely visible on some displays. Don't forget to take into account any computers you use as backups or off-season work, for instance your laptop computer. You can check the resolution on any computer by right-clicking on the Desktop and selecting Properties, then go to the Settings tab.

For Add-on dialogs, the Width & Height specify how much the original dialog is enlarged, if at all. You can enter 0 for either or both values if you don't need it enlarged for the changes you're making.

For non-add-on dialogs, the Width & Height specify the entire size of the dialog. This defaults to 600 x 400 by default, to make sure it fits on the smallest typical display (640x480, allowing for margins, task bars, etc.).

Quick-Add 'Save' & 'Cancel' buttons (non-Add-on only)

When you create a new dialog, you usually want to have Save and Cancel buttons on it for saving or cancelling the changes to the data fields. Clicking this button will automatically add Dialog Elements for Save and Cancel functions in the typical upper-right positions in the dialog. The elements added will include the Action expressions needed for typical functionality.

If you decide to change the size of the dialog after adding these, you can delete the previous Save and Cancel elements and then click this button again to add them in the correct position for the new size. (Or you could just edit the Left positions manually in the elements.)

If you add these before adding other Dialog Elements, then you might want to move them down to the bottom when you're done so they're the last things controls in the tab order (more on that later).

Save & Test Dialog

This function allows you to quickly test the dialog, or at least see the results of your modifications. This function can be used to test the dialog repeatedly without completely exiting the Edit function.

The appropriate host dialog will be opened (if it's an Add-on definition), or it will create and open your new dialog, using the first record of the appropriate Base Table type as the context record. This allows you to basically see where new items are positioned, what the data looks like, test its functionality, etc.

Be careful when testing, because any changes to the data in the dialog (or other functions activated) will be real changes!

Also be aware that it ***does*** completely save any changes you've made to the Dialog Definition as soon as you click the Save & Test button, so it negates any possibility of cancelling changes you've made to the definition.

Dialog Elements List

A list of Dialog Elements appears on the Edit Dialog Definition dialog. Dialog Elements are actually records linked to the Dialog Definition, so this list shows those linked records. As with most places where record lists are manipulated, the typical functions are available to Add, Insert, Edit, Copy, Delete, and Move items in the list. In addition, there are a few special functions as described below.

The list contains several fields of the elements, but depending on the element type there's no guarantee that each of these fields will be applicable. So using good Notes for the elements can help later.

Note that some columns (e.g. any expressions) may be truncated (with "..." at the end). This is done automatically to limit the column widths and keep long text from making the other columns hard to find. (Even if the columns or the whole dialog is enlarged, it won't show these fields any longer here.)

Dialog Element Order (& Tab Order)

When a Dialog Definition is invoked, the items are processed one after another in the order they appear in this list. While this in itself doesn't always make a difference, e.g. for "modification" or "action" element types, the order of New Controls in the list determines the order the controls are added to the dialog. This in turn affects the "Tab order", i.e. the order in which the fields are selected in the dialog (get input focus) when the Tab key is pressed. In other words, for text input fields the order determines the sequence that the cursor moves from one input field to the next.

In addition, the order of controls on the dialog is important for hot-key use. For instance if you define a label for a text field as "&Name", so that the "N" is an underlined hot key, then the text input control for that field must be the very next element in the list so that the cursor goes to that field when **Alt-N** is pressed.

Finally, the order is important for Radio type controls since it affects their grouping. This is covered in more detail later, under Editing Dialog Elements.

Quick-Add Fields

This is a special function used to easily add the necessary Dialog Elements for entering (or modifying) database fields. You will be able to select multiple fields, and it will automatically add the elements (with appropriate labels for input fields) positioned in a neat column on the dialog.

First you will need to specify several parameters for positioning the entry fields. The defaults will be set for typical positioning, starting in the upper left of the dialog. You can leave these defaults alone and Continue, at least for the first batch, if it's a new dialog. However if it's an Add-on, or if you've already added some elements, then you'll probably need to adjust the starting left and top position. You can also adjust the vertical spacing (to put multiple fields closer together or farther apart), as well as the left position and maximum width of the entry fields.

Next, a list will be shown with fields for the appropriate table(s) for the type of dialog you're adding to, or for the Base Table of a new dialog. Select as many fields as you want to add in one column of input fields. If you need to add multiple columns, e.g. if they won't all fit in one column, then you need to do it in separate sessions of Quick-Add Fields since this function won't automatically wrap to another column -- it will continue adding elements even if they go past the bottom of the dialog.

While it will do some auto-adjustment (restriction) of the positioning specified based on the dialog size, you may still end up with fields that go off the end of the dialog. You might also end up with overlapping controls, since no overlap-checking is done. These can be modified individually, or you can delete all of the new elements and try again.

Import and Export Elements

These functions will import or export any selected elements to a CSV (text) file. The main purpose of this is to copy the elements to a different Dialog Definition. You can also export commonly used sets of elements for importing to Dialog Definitions you create later. Basically it's like doing a copy/paste of elements, but going through a file instead of just the clipboard. Note that it's not the same as the import/export function for a complete Dialog Definition -- it's strictly for copying or moving selected elements within a Dialog definition.

When exporting, only the elements selected are exported (they don't need to be sequential, e.g. you can use Ctrl-click to select each element to export). When importing, all imported elements are added to the end of the definition.

Editing Dialog Elements

The Edit Dialog Element dialog is shown when adding or editing elements from the Edit Dialog Definition function.

Here you edit all of the components of a Dialog Element. This one dialog is used for all types of elements, even though different information is needed for each type. To help avoid confusion, it will only show the fields that apply to the element type that's currently chosen. Choosing a different Element Type will completely change most of the fields available on the dialog.

The common fields available to all elements are described first, followed by the type-specific fields for each element.

Element Type

Select the type of element you want to add. The dialog elements fall into several different categories:

Add a New Control -- This adds a new "control" to the dialog, such as an entry field, text, checkbox, selection list, etc. Naturally the control is only added once, when the dialog is created, but any Action expressions in the elements are still active as long as the dialog is open.

Modify a control -- Elements can be used to rename, move, resize or hide an existing control. These can only be done for standard controls on the dialog, not new controls that you add. Multiple elements can act on the same control, e.g. you can rename, move, and resize the same control using 3 different elements. These elements are executed only once, when the dialog is created.

Action on input -- Data-entry types of controls can trigger an action when something is changed (when the text is changed, a box checked, etc). You can define an expression to be executed when the action is triggered, e.g. to do something when a button is clicked, or auto-format text as it's entered. This element can only be added for standard controls, since the Add New Control element already contains an Action expression for new controls. Any action elements stay active for the duration of the dialog.

Action on focus change -- This can be used to trigger an expression when a control gains or loses input focus (e.g. due to a mouse click or Tab to the next control). These elements can be defined for either standard controls or for new controls you add.

Action on data saved -- This element is triggered when data is saved to the record being edited. This is usually done when "Save" is clicked on the dialog, but other things can trigger it too. This element can be used to do special validation of the data, for instance.

Action on dialog initialization -- This element is triggered when the dialog is being initialized. You can use this element to perform any other actions you want to when opening the dialog. Note that this is only

executed once, not for each record change (e.g. if changing records in Reservation Details), so the record context is not necessarily useful. For instance It could be used to do initial validations, modify controls, or show messages.

Condition

This optional expression determines whether or not the element is used, and must return a True or False boolean value. To edit the condition expression, click on the text box or click the Edit button next to it. The Expression Creator dialog will be used to enter the expression.

If the condition returns False, then the element is considered non-existent -- if it's for a New Control element, then the control won't be added. If it's for an element that modifies other controls, the modification won't take place. And if it's for an action type of element, then the action will not take place.

Note that for action elements, this condition is checked only when the dialog is first created, not each time there's a possible trigger, so a condition based on changes in the dialog cannot be checked here. If you want the element to only execute an action if certain changeable conditions are satisfied at the time of the input or focus trigger, then you need to check these conditions within the Action expression.

Notes

These notes are for your reference, and will also appear in the list of elements. We recommend putting notes in especially for action type elements if the expression isn't obvious, but it's a good idea for any element that's not obvious so you can tell what's going on if you need to edit the Dialog later.

"Add New Control" Elements

New control elements are used to add new data to a dialog, either to show the value or to allow the user to change a value. You can also add your own buttons to the dialogs, perhaps to perform special functions or to open a new dialog for entering a lot more information.

Control Name

This is a name to be used for the control internally, but is not the text shown for the control. A control name should be unique within the dialog definition, so that it can be referenced in expressions (e.g. to set or save the field's value) and referenced in other dialog elements. It may also help to indicate the type of control in the name for easy reference. Typical control names might be "EmailButton", "LastNameEntry", "LastNameLabel", "SiteDirtyCheckbox", etc. Using multiple words without spaces like this also serves as a reminder that the Control Name is not the text that gets displayed, but if you prefer to use spaces then you may do so.

Note that the names may be case sensitive when referenced (e.g. "ButtonA" is not the same as "buttona"), though this is not guaranteed. It's good practice to always assume the case must match.

Control Type

Select the type of control from the list. Naturally the type of control determines what's shown on the dialog as well as what it can do, how it acts, what kind of action it triggers, etc., so familiarity with Windows controls is helpful. A brief description of each is given below.

- Static text - Simple text (not editable), which will wrap as needed within the control's size
- Edit text - A single line text input field
- Edit text multi-line - A multi-line text input field
- Edit text, Read-only - Similar to static, but has a box around it like an edit field
- Edit text multi-line, Read-only - Same as above, for multiple lines.
- Button - A clickable button with text in it.
- Check box - A square checkbox input field for yes/no or true/false values
- Radio button - A round check-button field for multiple-choice input values
- Date selection - A date input field with a drop-down calendar selection
- Date selection with checkbox - A date field that has a checkbox (unchecked = no date selected)
- Time selection - A time entry field, with up/down buttons
- List box - A multi-line list of items to choose from
- Drop-down list box - A list of items to choose from, in drop-down form
- Combo list box - Similar to a drop-down list, but any text can also be entered

Start of a group

This is generally only important when Radio button controls are added, since a set of radio buttons must be in a "group" to act properly. In particular, when one radio button of a group is checked, it will automatically uncheck any others in the group. Also, it allows the arrow keys to be used to select which item in the group is checked, and the Tab key will move on to the next control outside the group.

Every new control has this option checked by default. When adding radio buttons for a multiple-choice option, you should **uncheck** this for each button except for the first one. It's also important that the next control in the dialog element list does have this option checked, so that it knows a new group (not part of the radio group) is starting.

Left & Top position

Specify the upper left corner position of the control, in pixel units. The position 0,0 is in the upper left corner of the dialog. It's usually desirable to leave a margin of about 10 pixels, so the default starting position is 10,10.

Width & Height

Specify the size of the control, in pixel units. Note that for drop-down list boxes and combo boxes, the height specified is the maximum height after it's dropped down -- the control height when it's not dropped down cannot be changed.

Control Text

This is the text that will appear in the control by default for text controls, buttons, check & radio boxes, and edit controls. It's not applicable to date or time controls, or list & combo boxes. These types need to be "filled in" from an Initialization expression, for instance.

Initialize Expression

This expression is executed for each control, after the control is created and shown on the dialog. This is where you should "populate" the value of the field if it's a data field. You can also use this to set static text fields to a value that depends on the context record, for instance. To edit the expression, click on the text box or click the Edit button next to it. The Expression Creator dialog will be used to enter the expression.

There are **DlgSetUserCtrl...( )** functions for each type of control, so you should use the appropriate expression to set the value of the control. Here's an example, setting an Edit control to the current value of the "# Adults" reservation field:

```
DlgSetUserCtrlText("# Adults Input", FieldText(ThisResv(),"Resv_Adult"))
```

For list and combo box fields, you'll also need to populate the list of selections, using **DlgAddUser...( )** function calls. Here's part of an example for a Site Type list, including a blank entry at the top (assuming you want the option to leave it blank). Here the **Eval( )** function is used so that many expressions can be executed in sequence, in order to add all of the necessary options:

```
Eval( DlgAddUserDropListText("Type Input"," "), DlgAddUserDropListText("Type
Input","Normal RV"), ....
```

If you're not sure how to handle a particular type of input, use the Quick-Add Fields function in Edit Dialog Definition to add a similar type of field. It will fill in an appropriate expression for you.

OK/Save Expression

This expression is executed for each control when the "Save" function is done for the dialog (e.g. when **DlgClickSave( )** is called in a new dialog, usually done when the "Save" button is clicked). This is where you would get the final value of the control and put it into the data field of the appropriate record. To edit the expression, click on the text box or click the Edit button next to it. The Expression Creator dialog will be used to enter the expression.

It may be tempting to do validation here, but there's no way to abort the saving here if the value isn't valid. See the other element type, "Action on data Saved", for a good place to do validation. Of course if you just need to modify the value, e.g. convert the typed text to upper case, then you can do that here.

You should use appropriate expression functions to get the value of the control. There are **DlgGetUserCtrl...( )** functions for each type of control. Here's an example, getting an Edit field for the "# Adults" reservation field and setting the data value:

```
SetFieldText( ThisResv(), "Resv_Adult", DlgGetUserCtrlText("# Adults Input"))
```

If you're not sure how to handle a particular type of input, use the Quick-Add Fields function in Edit Dialog Definition to add a similar type of field. It will fill in an appropriate expression for you.

Action Expression

This is optional expression which will be executed when the control's value is changed by the user (e.g. a button or checkbox clicked, text edited, or a list selection changed. As with any expression, appropriate context will be available. To edit the action expression, click on the text box or click the Edit button next to it. The Expression Creator dialog will be used to enter the expression.

"Modify Control" Elements

These element types, Rename, Move, Resize and Hide, allow you to change the standard controls of an existing dialog. These cannot be used with new controls you add, since all of this can be specified in the new control's element definition already.

Control ID

This specifies which dialog control will be modified. It must be entered as the internal ID of the control. To find out what the ID is for a given control, you will probably need to contact the software provider. You can also use a tool like the Spy++ program that comes with Microsoft development systems, if you have such a tool available.

Position, Size, or Control Text

Depending on the type of element, fill in the appropriate information -- the new Left and Right position for moving a control, or the new Width and Height for resizing a control, or the new Control Text for renaming a control.

"Action on Input" Elements

These element types allow you to perform some action when the standard controls of an existing dialog are clicked or the values changed (depending on the control type). These types of elements cannot be used with new controls you add, since the action expression is specified in the new control's element definition already.

Note that "Action on button click" is used not only for button controls, but also for checkbox controls and radio buttons. If the wrong type of action element is used for a control (e.g. a "button click" action for an Edit control), the results are unpredictable.

Only one Action element for a given control will be processed at the time of the input trigger (based on the order in the Dialog Elements list). You can define more than one action element for a control, using the Condition expression to determine which one(s) will be used. But if more than one action element for the same control meets the conditions and is "used", only the first one will be executed.

If you need to do more than one thing or check for dynamic conditions at the time of the input trigger, then the Action Expression should handle everything in one expression.

Control ID

This specifies which dialog control will be modified. (See "Modify Control" Elements above for details.)

Action Expression

This is the expression which will be executed when the appropriate control change is triggered. As with any expression, appropriate context will be available. To edit the action expression, click on the text box or click the Edit button next to it. The Expression Creator dialog will be used to enter the expression.

Execute action before normal processing

If this option is checked (and an Action expression entered), the Action expression will be executed before any normal processing that might happen for the action. This may or may not be important, depending on what you want to do.

The "normal processing" refers to the internal processing that the program normally does when this action is triggered. In many cases it's nothing, e.g. when entering text for most fields. However in many cases there is something done such as validation of the value, or changing what other fields are visible. And of course any buttons clicked will do something important.

As an example, you might define an element to execute an action when the "+1M" button is clicked in Reservation Details. If you need to make sure it's OK to do this before allowing it, then you would check this "Execute action before..." option so you can do your check and keep the function from happening if needed). However if you want the normal action to take place first (updating the date) before executing your expression, then don't select this option.

Abort the operation which triggered this action

If this is checked, then the Action Expression should return a True or False (boolean) value. If the value is False, then any normal processing is aborted. Obviously this is only useful if you also check the option to execute your action before normal processing, since there's nothing to abort after the normal processing.

As an example, you can do some validation or prompt the user when a button is clicked (e.g. to Check In a reservation), which would naturally need to be done before the normal processing, and return False in your Action expression to abort the operation if you don't want it to continue.

"Action on Focus Change" Elements

These elements can be used to trigger an expression when a control gains or loses input focus (e.g. due to a mouse click on the control, or the **Tab** key pressed to get to the next control). These elements can be defined for either standard controls or for new controls you add.

As described for "Action on Input" elements, only the first applicable focus change element will be used for a given dialog control.

Control ID or Control Name

This specifies which control will be modified. If it's for a standard (existing) control, fill in the Control ID (see "Modify Control" elements above for details.) If it's for a new control, fill in the Control Name of the new control (which must match the Control Name in the new control element definition).

Action Expression

This is the expression which will be executed when the control loses or gains focus. To edit the action expression, click on the text box or click the Edit button next to it. The Expression Creator dialog will be used to enter the expression. The expression is always executed after the focus change actually happens (so for instance you can set focus somewhere else).

The focus change cannot be aborted, so the return value of this expression is not used.

"Action on Data Saved" Elements

For add-on Dialog Definitions, this element is triggered when data is saved to the record being edited.

For new dialogs, this is triggered when the **DlgClickSave( )** function is executed. This is usually done in the Action expression for the "Save" button, so it will happen when "Save" is clicked on the dialog.

Action Expression

This is the expression which will be executed when the data is saved to the record being edited. Note that for add-on dialogs this isn't necessarily just when the dialog is closed -- for instance when used in the Customer Details dialog (Edit Customer), this is done whenever the customer being viewed is changed also, e.g. through the Next and Prev buttons.

To edit the action expression, click on the text box or click the Edit button next to it. The Expression Creator dialog will be used to enter the expression.

Execute action before normal processing

If this option is checked, the Action expression will be executed before any normal processing that the dialog does for saving -- generally this means before the data is actually saved into the record. Thus you can do any validation of your own before anything is saved, with the option to abort the changes before actually saving them.

If this option is not checked, then the Action expression is not executed until all of the normal "Save" processing is performed by the dialog, e.g. the data validated and saved.

Note that it is possible to have two "Action on Data Save" elements, one to be executed before and one to be executed after the normal processing.

Abort the operation which triggered this action

If this is checked, then the Action Expression should return a True or False (boolean) value. If the value is False, then any normal processing (validation and saving data) is aborted. Obviously this is only useful if you also check the option to execute your action before normal processing.

Event Actions

Overview

Event Action definitions allow you to "inject" functionality into Campground Master by trapping, or intercepting, a specific change or activity in the program. There are many pre-defined "Events" in the program that can be intercepted and acted upon, and you can conditionally execute any expression in the context of that event. In many cases you can not only perform an action but also stop the event from completing. For instance, you can check new reservations for meeting certain criteria like minimum stays on holiday weekends, and not allow it to be saved unless it meets this criteria.

Most reservation status changes are trappable events, as well as some customer and transaction changes and dialogs. There are also some general actions like operator log-in and log-out, program start and stop, and timer events to allow an action to be performed every second, minute, hour or day.

As with all advanced customizations, this requires a rather technical understanding of the system. **However it must also be emphasized that Event Actions are much more "dangerous" than any other customizations because they define actions that happen automatically, potentially changing the overall operation of the program, and also potentially causing a serious system malfunction.**

Event Actions Setup

To create an Event Action, go to Maintenance / Advanced Customizations / Event Actions. This opens the Event Actions Setup dialog, which lists all current Event Actions and has the typical functions for Adding, Inserting, Editing, etc.

You can also Export one or more Event Actions to a text file, or Import Event Actions. This is primarily for you to import Actions created by the software provider, though it can also be used to transfer Actions between multiple databases.

Event Actions cannot have duplicate names. If you make a Copy of an Action, text like "(copy 1)" will be added to the name to make sure it's unique. Of course you can change this to be more appropriate. Duplicate checking for the names is not case-sensitive ("My Action" is considered the same as "my action").

Rearranging Event Actions

The order in which the Event Actions appear here in Event Actions Setup will determine the order in which they're processed, in particular if there's more than one Action defined for the same Event Trigger. While this is not likely to matter in most cases, if more than one Action is triggered by the same Event, then the results of one could affect other. If you want to move them around, use the Move Up and Move Down buttons.

Editing Event Actions

The Edit Event Action dialog is shown when adding or editing an Action from Event Actions Setup.

Action Name

The name should be descriptive enough for selecting the Action form the Setup list -- the name is not used anywhere else. Each Action must have a unique name (which is not case-sensitive).

Enabled

The Action can be disabled so that it does not get triggered regardless of the conditions. Technically you could also delete the Action, but if you might want to use it later or keep it for reference then it's better to just disable it. This is also handy for disabling Actions you haven't finished or tested sufficiently.

Note that Event Actions don't have an access level setting (presumably most actions should happen no matter who is logged in) -- if an action should depend on an access level, then include an expression to check the level of the current operator in the Condition (see below).

Event Trigger

Select the event from the list that you want to trigger your action. Note that most events have both a "before" and "after" option, so for instance you can do an action before and/or after the event is processed internally. As an example, you can have an action that does some validation before a reservation is checked in (optionally preventing the check-in with the Abort option below). You could also have an action that's triggered after the check-in is complete, such as printing a form (which would not get triggered if the check-in was aborted).

Note that there are some grey areas as far as exactly when the action will be triggered, which means the internal sequence is not strictly defined and could change in future versions. For instance an after-check-in action is guaranteed to be triggered after the "Checked In" status is changed for a reservation, but it whether it's triggered before or after the charges and payments are applied, a receipt is printed, and other synchronized reservations are updated is all grey area -- so try not to assume anything about related changes.

While there are events for viewing customer and reservation details and transactions, don't forget that the Dialog customizations can also allow expressions to be executed when entering or exiting these dialogs (after the dialog is opened and before it's closed, as opposed to before it's opened and after it's closed). The exact nature of what you need to do might make using one or the other of these customizations more appropriate.

Time Triggers

There are triggers for "Every Second", "Every Minute", "Every Hour" and "Every Day". They sound self-explanatory enough, but there are several caveats to these.

Generally these are triggered at the "change" of the minute, hour, and day (when the system first notices that it has changed, e.g. the "Every Hour" trigger will happen as soon as the hour digit changes on the clock).

However, timed events are suspended when any dialog is open so that the operator is not interrupted (and because the data could be in an inconsistent state while a dialog is open). Once all dialogs are closed, then event checking is resumed. For "Every Second" events, this generally means that all intervening triggers are lost (e.g. it won't try to catch up with possibly 100's of "missed" triggers). However it does check against the last time an event was actually triggered, so for instance if the Minute or Hour rolled over while a dialog was open, then that event will immediately be triggered as the change is noticed. Of course this also means that the event is not guaranteed to happen "exactly" at the roll-over time.

It can be difficult to get events to happen at regular intervals with any consistency. For instance if you want something done every 30 minutes, one method would be to define an "Every Minute" trigger, and use the Condition expression to check for the current time's Minutes being either 0 or 30. However, if a dialog is open for more than a minute, e.g. from 3:59 to 4:01, then the event completely misses the 4:00 check and the condition won't be met until 4:30 (assuming a dialog is not open then also...). One way around this is to create your own Setting that stores the last time your Condition expression was executed, and check for that being 30 minutes ago (or longer). Then you don't rely on the exact time the event is triggered, but it's still subject to how long a dialog is left open, so intervals would commonly vary from 30 minutes to perhaps 45 minutes or more, depending on how long an operator works on something.

Important Note: It may be tempting to do something like this Every Second to make sure it's checked as often as possible. However you don't want to execute a complex expression every second, since it could significantly lock up the system, so be conservative on how often something needs to be done. For instance, setting the value of a Setting can take significant time because it involves a database change (which means immediately saving the database to the hard disk, refreshing views, synchronizing other workstations, etc.), which could completely lock up the system. (In fact it would be more efficient to use a file to read & write the last-checked time, since it won't affect anything else and is much faster than writing the whole database.)

Condition

This expression is executed when the action is triggered, and the result determines whether the Action is executed. Naturally the result should be a boolean (True or False) value, or else leave the expression blank to always execute the Action. If the event involves a particular record, e.g. a Reservation, then the appropriate context information will be available to the expression.

To edit the condition expression, click on the text box or click the Edit button next to it. The Expression Creator dialog will be used to enter the expression.

Action

This is the expression to execute for the event (if the Condition is met, of course). If the event involves a particular record, e.g. a Reservation being checked in, then the appropriate context information will be available to the expression. The result of the expression is ignored and can be any type of value unless the Abort option is selected (below).

To edit the action expression, click on the text box or click the Edit button next to it. The Expression Creator dialog will be used to enter the expression.

Abort the Event....

If this option is selected, then the result of the Action expression is checked for a False (**.F.**) value (the result of the Action expression should be a boolean value, of course). If it's False, then any further processing for the event which triggered this action will be aborted. Obviously this is usually only useful for a "before" type of event trigger, since the "after" triggers only happen after all of the important stuff has already happened. However if you define more than one "after" action for the same event, then aborting will stop any following Actions from being processed.

This is generally used to check some special condition before continuing with an event like checking in, checking out, etc. You could even open a special dialog or prompt for the user (e.g. "Is the customer over 18 years old?"), and decide whether or not to continue based on the results.

Note: It's usually a good idea to include a **MessageBox( )** function in the Action expression to show a message if you're going to abort further processing, so that the user knows that the event is going to be aborted (and why).

Import Package

This function, available through Maintenance / Advanced Customizations / Import Package, is primarily for you to import customizations created by the software provider. This difference between this function and the "Import" functions available on the various Setup dialogs (e.g. Setup Forms) is that this function will import every record in the selected file no matter what kind it is -- so it can import Forms, Dialogs, Menus, Color Schemes, etc in one step instead of going to each function separately.

When you select this function you'll get a typical Windows file dialog labelled "Import Package". You need to locate the appropriate folder which contains the package file, select the appropriate file, and click Open. Once the package is imported, nothing in particular will indicate success -- however there may be a warning shown due to duplicate names imported.

Note that the import/export files use the "CSV" file extension (e.g. "Sample.csv"), which means it's a comma-separated-value text file. Windows may recognize this file as something another program can open, but these are in a special format for Campground Master and should not be used in other programs.