1
The Basic Ingredients

> He who has not first laid his foundations may be able with great ability to lay them afterwards, but they will be laid with trouble to the architect and danger to the building—Niccolo Machiavelli
>
> To me programming is more than an important practical art. It is also a gigantic undertaking in the foundations of knowledge—Grace Murray Hopper

In Chapter 1, we will deal with the basic foundations of Microsoft Dynamics NAV (pronounced as N-A-V, spelling it out), the objects that make up an NAV application, and their essential capabilities and limitations. While NAV has many structural and syntactical similarities to other programming languages, particularly Object Pascal; NAV has many unique features and facilities as well.

Once you are through with the Chapter 1, you will feel more comfortable with the NAV development environment, will get acquainted with the tools, and will look forward to get more detail. Also, you will develop knowledge that will allow you to begin thinking about the application development within the NAV environment, using the NAV programming language.

While learning NAV development environment, we will develop a simple application as a functional enhancement to the base product. Our application will be designed for the management of a fictitious association for those who work with **C/SIDE** and **C/AL**. We'll call it the worldwide Charter/Association of NAV Developers Ltd, or **C/ANDL** for short. The goal of C/ANDL is to shed a little light into NAV Development. We will deal with member records, skills and education information, hold meetings, and offer some training on publications for sale. Our application will be designed as a new application function, but with the plan of using base functionality for various accounting functions.

Some Unique NAV Terms Defined

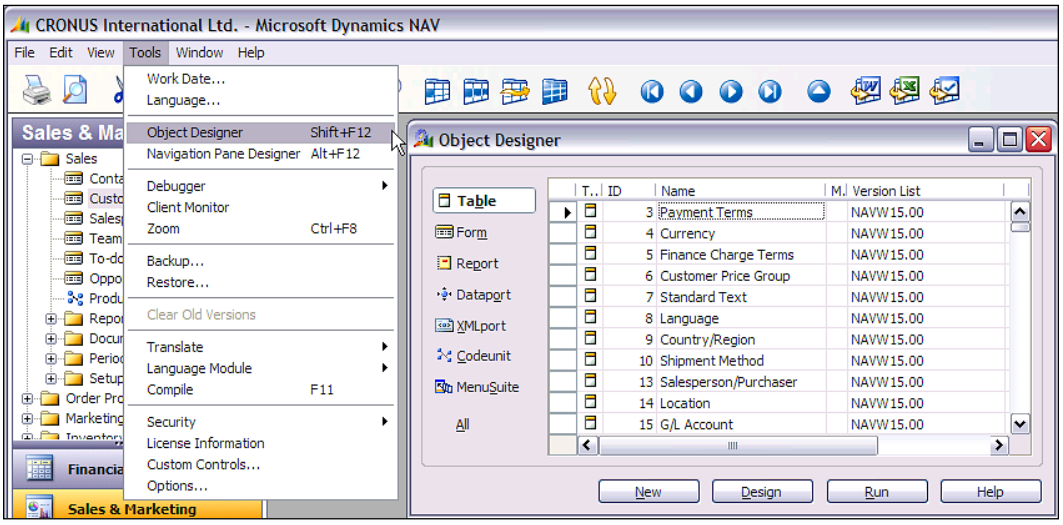
The following are some unique definitions in NAV:

- **C/AL**: Client/Application Language is the programming language meant for customization of NAV. It was built by the NAV development team using C++, though we never see any C++ code directly in the NAV product. C/AL is the tool used to define the processes by which data is manipulated, to define the business rules that will control the various applications, and to control the flow of all the logical processing sequences. C/AL is also used to manipulate objects, to control the execution flow of objects, to create new functions complementing the functions that are built-in, and to manipulate data in many different ways.
- **C/SIDE**: Client/Server Integrated Development Environment is the development tool specified for using C/AL. It includes the language editor, compiler, debugger, reports, form generators (called designers in NAV), and code management tools. Almost all the C/AL development, along with NAV development, is done within C/SIDE without the use of external tools. For most application development, NAV is entirely self sufficient except for those services provided by the Windows operating systems. It is possible, though generally not recommended, to write code using a text editor and then import it into C/SIDE.
- **Filtering**: The application of range constraints is to control what data is processed or made visible. For example, a filter for payment data for Customer No. 20134 would show the payments for that customer only. Although not really unique to NAV, filters combined with other NAV features are uniquely powerful in NAV. The extreme flexibility of filtering in NAV allows you to easily create very focused views into the data. Filters can be defined as ranges, boolean expressions, specific selections, etc. which delimits the data to be selected into a subset to be utilized in a process (display, calculation, report, etc.). Thus, NAV filters are a very powerful tool for both the developer and the user.
- **SIFT**: Sum Index Field Technology is a very clever method of providing instantaneous response to user inquiries. Most application systems provide fast response to requests for summary information by maintaining pre-calculated total ("bucketed data"). NAV retains all data in detail and, through the use of SIFT and applied data filters; it provides the activity totals or subsets of information subject to a wide range of selection constraints instantly. Your data structure design will determine whether SIFT results are available and if available, how fast the response will be. Even though the designers of NAV were very clever in giving you special tools to use, you are still responsible for how well those tools will work for your users.

- **C/FRONT**: It is an application programming interface that allows you to develop applications in other programming languages to access a Microsoft NAV database, either the C/SIDE Database Server or the Microsoft SQL Server. The primary component of C/FRONT is a library of callable C functions which provide access to every aspect of data storage and maintenance. This allows creation of custom components written in C, C++, VB, Delphi, and the Visual Studio.NET languages as well as other languages which support compatible calling conventions. C/FRONT is only tested by Microsoft for use with code built using either the Watcom C or Microsoft C++ compilers. C/AL triggers cannot be invoked via C/FRONT code. C/FRONT comes as a set of files to be installed guided by the instructions given in the C/FRONT manual.
- **C/OCX**: It is an application interface to allow integration between C/AL and a properly defined OCX routine. This allows access to many ActiveX controls available from third party vendors. Such controls must be non-visual as far as NAV is concerned (but they may open their own windows for user interaction).

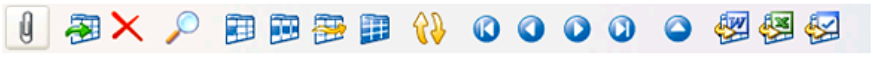
The C/SIDE Integrated Development Environment

The C/SIDE Integrated Development Environment is referred as the **Object Designer** within NAV. It is accessed through the **Tools | Object Designer** menu option as shown in the following screenshot:



Object Designer Tool Icons

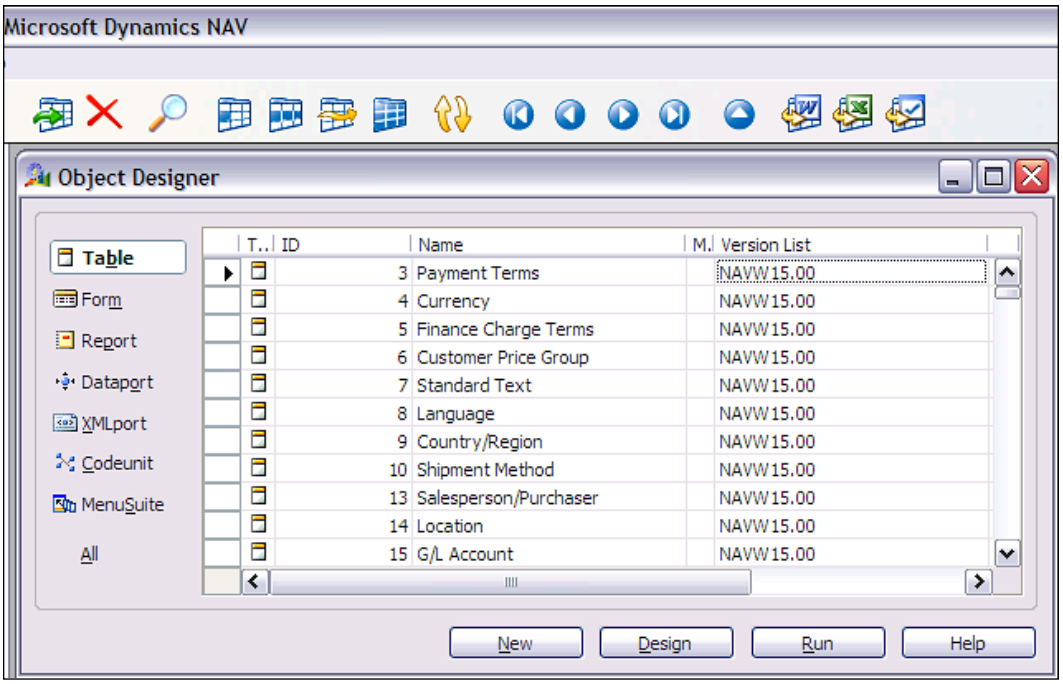
The following screenshot shows **Object Designer** form, containing a list of several tool icons. These Object Designer **tool Icons** are shown isolated in the screenshot and then described briefly in the following table. Some of the terminologies in these descriptions will be later explained in this book. Additional information is available in the C/SIDE **Help** files and the Microsoft NAV documentation.



Icon	Name	Keys	Description
	Links	Ctrl+L	Links a NAV record to a document, a folder, a web site or another NAV record.
	New	F3	Open up a new record entry
	Delete	F4	Delete a record
	Find	Ctrl+F	Invoke the search capability
	Field Filter	F7	Enter a filter on the highlighted field
	Table Filter	Ctrl+F7	Enter (or edit) one or more filters on a table
	Flow Filter	Shift+F7	Enter, edit or remove a flow filter
	Show All	Shift+Ctrl+F7	Clear all filters, except flow filters
	Sort	Shift+F8	Display the active keys so that a new sort order may be chosen
	First	Ctrl+Home	Jump to the beginning of the table
	Previous	Down Arrow	Move to the Previous record
	Next	Up Arrow	Move to the Next record
	Last	Ctrl+End	Jump to the end of the table
	List	F5	Displays the defined List form
	Send to Microsoft Office Word	Ctrl+W	Export current record data to a Word document using the last previously used Style Sheet for this form
	Send to Microsoft Office Excel	Ctrl+E	Export the data in the current record to an Excel document using the last previously used Style Sheet for this form
	Send Options		Display the defined XML export targets and Style Sheets for the current record.

Seven Kinds of NAV Objects

NAV C/AL is not considered an object oriented language even though C/AL uses seven kinds of objects. These seven object types are listed on the left side of the **Object Designer** window as shown in the following screenshot:



NAV is not an object oriented language because you can only use the predefined object types. The seven types of objects in C/AL are as follows:

- Table: The definers and containers of data.
- Form: The screen display constructs for user interface.
- Report: It allows the display of data to user in "hardcopy" format, either onscreen (preview mode) or via a printer device. Report objects can also update data in processes with or without accompanying data display output to the user.
- Dataport: It allows the importing and exporting of data from/to external files.
- XMLport: It is similar to **Dataport** but specific to only XML files and XML formatted data.
- Codeunit: They are the containers for code.
- MenuSuite: It consist of menus, structured differently from other objects.

More Definitions (Related to NAV)

The following are the few more definitions related to NAV:

- **Database**: It consists of two database definitions (physical and logical) and two implementations (C/SIDE Database Server and Microsoft SQL Server). The C/SIDE Database Server was formerly known as the "Native" database because for number of years, this proprietary NAV that supplied database was the only database for NAV. In earlier versions, it did not have a name other than "the NAV (or Navision) Server". The logical database definition relates to the sum total of the relationships between data, the indexes that control data access, and in NAV, the **SumIndexFields** and **FlowFields** (refers to special data summing features, these terms are explained in detail in a later Chapter).

  NAV is a relational database system. The Development Environment (C/SIDE) and tools (C/AL), makes the choice of underlying database (C/SIDE Database Server or Microsoft SQL Server) platform almost transparent to the developer. In this book we will not concern ourselves with the physical database definition because, except in rare circumstances, our development work will not be guided by physical database factors. When you become involved in more complex design activities, you will likely need to be concerned about the differences between the two database options.
- **Properties**: These are the attributes of the element (e.g. object, data field, or control) which defines some aspect of its behavior or use. For example, display length, font type or size, and the elements are either editable or viewable.
- **Fields**: They are the individual data items.
- **Records**: These are the group of fields (data items) that are handled as a unit in most Input/Output operations. The table data consists of rows of records and columns consisting of fields.
- **Controls**: They are the containers for constants and data. The visible displays in reports and forms consist primarily of controls.
- **Triggers**: The generic definition is a mechanism that initiates (fires) an action when an event occurs such as reaching a certain time or date or upon receiving some type of input. A trigger generally causes a program routine to be executed. NAV triggers have some similarities to those in SQL, but they are not the same. NAV triggers are locations within the various objects where a developer can place comments or C/AL code. The following are the NAV triggers:
  - **Documentation Triggers** consist of comments only. Every object type except MenuSuite has a single Documentation trigger.
  - **Event Triggers** are "fired" when the specified event occurs. Each object type has its own set of predefined triggers. The event trigger name begins with the word "On" such as `OnInsert`, `OnOpenForm` and `OnNextRecord`.

  - **Function Triggers** are the "functions" defined by developer. They represent callable routines that can be accessed from other C/AL code either within or outside the object where the called function resides. Many function triggers are provided as part of the standard product. As a developer, you may add your own custom function triggers as needed.

- **License**: A data file supplied by Microsoft that allows a specific level of access to specific object number ranges. NAV licenses are very clever constructs, which allow distribution of a complete system, all objects, modules, and features while constraining exactly what is accessible and how it can be accessed. Of course, each license feature allowing access to various objects and system functions, including the ability to do development, has its price. Microsoft Partners have access to licenses to provide support and customization services for their clients. The broadly featured Partner licenses are often referred to as developer's license, but end user firms can also purchase licenses allowing them developer access to NAV.
- **Object numbers and field numbers**: The object and field numbers from 1 (one) to 50,000 and in the 99,000,000 (i.e. 99 million) range are reserved for use by NAV as part of the base product. Objects in this number range can be modified or deleted, but not created with a developer's license. Field numbers are often assigned in ranges matching the related object numbers (i.e. starting with 1 for product fields in objects numbered 1 to 50,000, starting with 99,000,000 for fields in objects in the 99,000,000 and up number range).

  Object and field numbers from 50,001 to 99,999 are generally available to us for assignment as part of an ad hoc customization developed in the field using a normal development license. But object numbers from 90,000 to 99,999 should not be used for permanent objects as those numbers are often used in training materials. Microsoft allocates other ranges of object and field numbers to ISV (Independent Software Vendor) developers for their add-on enhancements. Some of these (in the 14,000,000 range in North America, other ranges for other geographic regions) can be accessed, modified, or deleted but not created, using a normal development license. Others (such as in the 37,000,000 range) can be executed but not viewed or modified with a typical development license. The following table summarizes the content as:

Object Number Range	Usage
1 – 9,999	Base application objects
10,000 – 49,999	Country specific objects
50,000 – 99,999	Customer specific objects
100,000 – 99,999,999	Partner created objects

- **Work Date**: It is a date controlled by the operator that is used as the default date for many transaction entries. The System Date is the date recognized by Windows. The work date can be adjusted at any time by the user, is specific to the workstation and can be set to any point in the future or the past. This is very convenient for procedures such as closing off sales order entry for one day at 5:00 pm, and then having the second shift sales order dated to the next calendar day. You can set the work date by selecting **Tools | Work Date**, and then entering a date.

NAV Functional Terminology

For various application functions, NAV uses terminology that is more akin to accounting terms than to traditional data processing terminology. Some examples are as follows:

- Journal: A table of transaction entries, each of which represents an event, an entity, or an action to be processed. There are General Journals for general accounting entries, Item Journals for changes in inventory, etc.
- Ledger: A detailed history of transaction entries which have been processed. For example, General Ledger, a Customer Ledger, a Vendor Ledger, an Item Ledger, etc. Some Ledgers have subordinate detail ledgers, typically providing a greater level of date plus quantity and/or value detail.
- Posting: The process by which entries in a Journal are validated, and then entered into one or more Ledgers.
- Batch: A group of one or more Journal entries that were Posted in one group.
- Register: An audit trail showing a history by Entry No. ranges of the Journal Batches that have been Posted.
- Document: A formatted report such as an Invoice, a Purchase Order or a Check, typically one page for each primary transaction.

Getting Started with Application Design

Our design for the C/ANDL application will start with the beginning of a Member Table, a Member Card, a Member List Form, and a Member List Report. Along the way we will review the basics of each of the NAV object types.

Tables

Table objects are the foundation of every NAV application. Every project should be started by designing the tables. Tables contain the definitions of the data structures, the data relationships within and between the tables, as well as many of the data constraints and validations. The coded logic in table triggers not only provides the basic control on the insertion, modification and deletion of records, but also embodies many of the business rules of an application. As we see when we dig into tables further, such logic isn't just at the record level but also at the field level. Putting as much of an application design as possible within the tables makes the application easier to develop, debug, support, modify, and upgrade.

Example: Table Design

Let us try a simple introduction to creation of a table for our NAV Developer Association application. We will create a basic `Member` Table which contains the following data fields: The first thing we will do is inspect the existing definitions for tables containing name and address information, such as the `Customer` table (table object 18) and the `Vendor` table (table object 23). From the common definitions in the following table, we see some patterns as to field names and definitions that we decide to copy.

Field names	Definitions
Member ID	10 character text (code)
Title / Prefix	10 character text
First Name	20 character text
Middle Initial	3 character text
Last Name	20 character text
Suffix	10 character text
Address	30 character text
Address 2	30 character text
City	30 character text
State/Province	10 character text
Post code	20 character text (code)
Country/Region code	10 character text (code)

We will not be using all these data fields for the `Member` table, but they give a reasonable start. This is a good illustration of how the design must begin with the tables. As you can see, at the preceding data field list, three of the fields have special text formats. That is because these are going to be referenced by or will reference to other data tables, so these are the data codes rather than descriptive data.

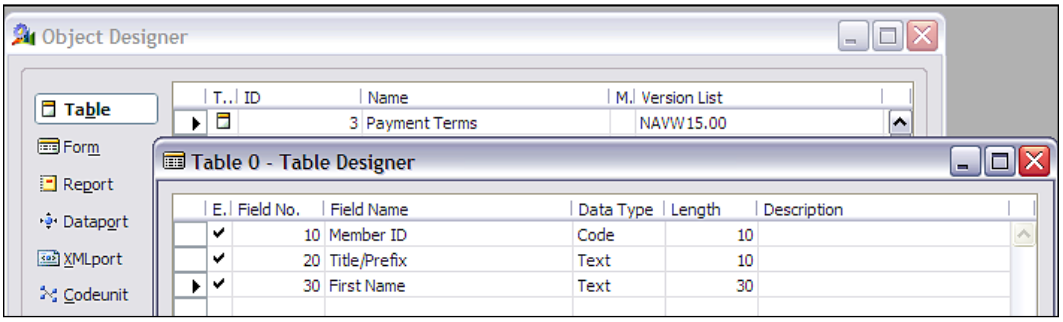
The **Member ID** will be the unique identifier for our Member record as it will also be referenced by other subordinate tables. The **Post code** and **Country/Region code** will reference other existing tables for validation. We choose the name, size, and data definition of these last two fields based on inspecting the equivalent field definitions in the `Customer` and `Vendor` tables.

We will have to design and define any referenced validation tables before we can eventually complete the definition of the `Member` Table. But our goal at the moment is just to get started.

Example: Table Creation

Open the **Object Designer**, click on **Table** (on the left column of buttons) and click on **New** (on the bottom row of buttons). Enter the first field name (**Member ID**) in the **Field Name** column and then, enter the data type in the **Data Type** column. For those data types where length is appropriate, enter the maximum length in the **Length** column. Enter **Description** data as desired; these are only for display here as internal documentation.

As you can see in the following screenshot (and will have noticed already if you are following along in your system), when you enter a Text data type, the field length will default to 30 characters. This is simply an 'ease of use' default which you should override as appropriate for your design. The 30 character **Text** default and 10 character **Code** default are used because this matches many standard application data fields of those data types.



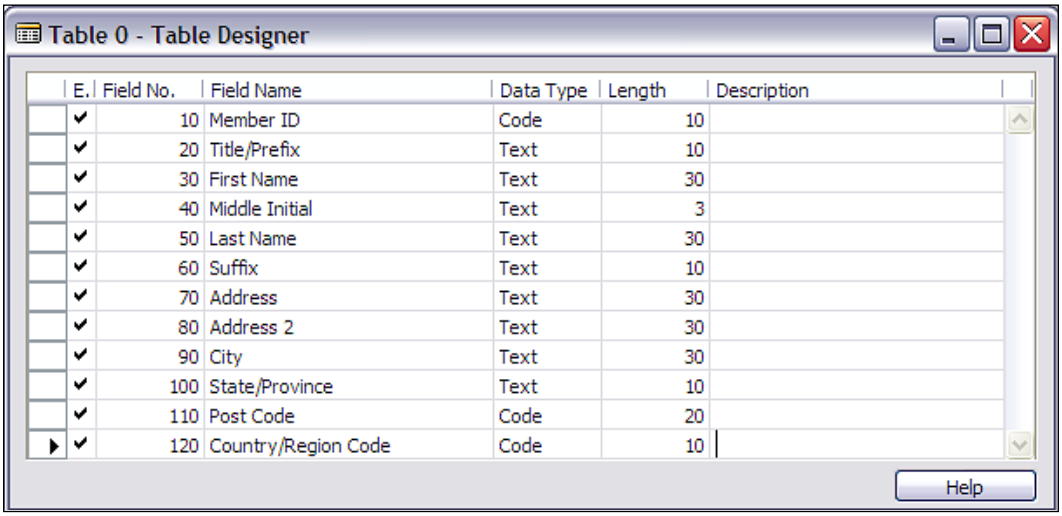
The question often arises as to what field numbering scheme to use. Various systems follow a variety of standard practices. In one system you might increment the field by two's, in another by five's, and in another by 1000's but in NAV, when you are creating a new table from scratch, it is a good idea to increment the **Field No.** by 10 as you have seen in the above screenshot. The default increment for Field No. is 1. For a group of fields (such as an address block) where you are certain you will never add any intervening fields, you could leave the increment at 1. But, there is no penalty or cost for using the larger increment, so it's not a bad thing to do all the time.

The numeric sequence of fields determines the default sequence in which data fields will display in a wide variety of situations. An example would be the order of the fields in any list presented to the user for setting up the data filters. This default sequence can only be changed by renumbering the fields. The compiler references each field by its **Field No**. not by its **Field Name**, the renumbering of fields can be a challenge once you have created other routines that reference back to these fields. At that point, it is generally better to simply add new fields where you can fit them without any renumbering.

In fact, it can be irritatingly painful to renumber fields at any point after a table has been defined and saved. In addition to the field numbers controlling the sequence of presentation of fields, the field numbers control bulk data transfer (those transfers that operate at the record level rather than explicitly field to field transfer—e.g. the TRANSFERFIELD instruction). In a record level transfer, data is transferred from each field in the source record to the field of the same number in the target record.

So you can see that it is a good idea to define an overall standard for field numbering as you start. Doing so makes it easier to plan your field numbering scheme for each table. Before you begin, enter the definition into C/SIDE. Your design will be clearer for you and your user, if you are methodical about your design planning before you begin writing code (i.e. try to avoid the Ready-Fire-Aim school of system development). The increment of **Field No.** by 10 allows you to insert new fields in their logical sequence as the design matures. While it is not required to have the data fields appear in any particular order, but it is frequently convenient for testing and often clarifies some of the user interactions.

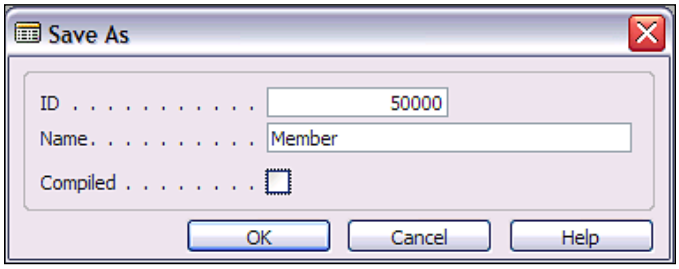
When you have completed this first table, your definition should be like the following screenshot:



At this point, you can exit and save your `Member` Table. The easiest way to do this is to simply press *Esc* until you are asked to save your changes. When you respond by clicking **Yes**, you will be asked for the Object Number and Name you wish to assign. In a normal development situation, you will want to plan ahead what Object Number and descriptive Object Name you want to use. We will discuss Object Numbering in more detail later. In this case, we will use table Object No. 50000 and name it as Member. We are using 50000 as our Table Number just because it is the first (lowest) number available to us for a custom table through our Table Designer granule license.

Note that NAV likes to compile any object as it is saved, so the **Compiled** option is automatically check marked. A compiled object is one that can be executed. If the object we were working on was not ready to compile without error, we could unselect the **Compiled** option in the **Save As** window as shown in the following screenshot.

Be careful, as uncompiled objects will not be considered by C/SIDE when changes are made to other objects. Until you have compiled an object, it is a "work in progress", not an operable routine. As a matter of good work habits, make sure that all the objects get compiled before you end work for the day.



Forms

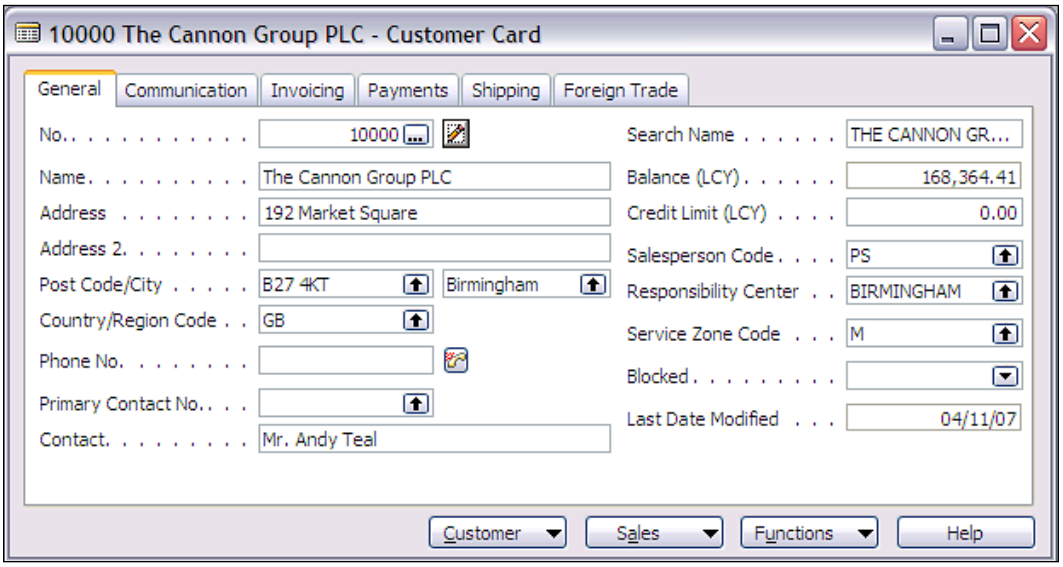
Forms fulfill two basic purposes. Firstly, it provide views of data or processes designed for on-screen display only. Secondly, it provides a key point of user data entry into the system. In standard NAV, there are two basic types of forms:

- Card forms
- Tabular forms

From a practical point of view, there are also special versions of card forms which use a Matrix control and therefore often referred to as Matrix form. Beyond that, there is a variation of the Matrix form, which is called as Trendscape form. There are also combination forms consisting of a Card form plus a Tabular form, called a **Main/Sub Form**. These are the user interfaces that appear as forms but are not form objects. These user interfaces use various dialog functions.

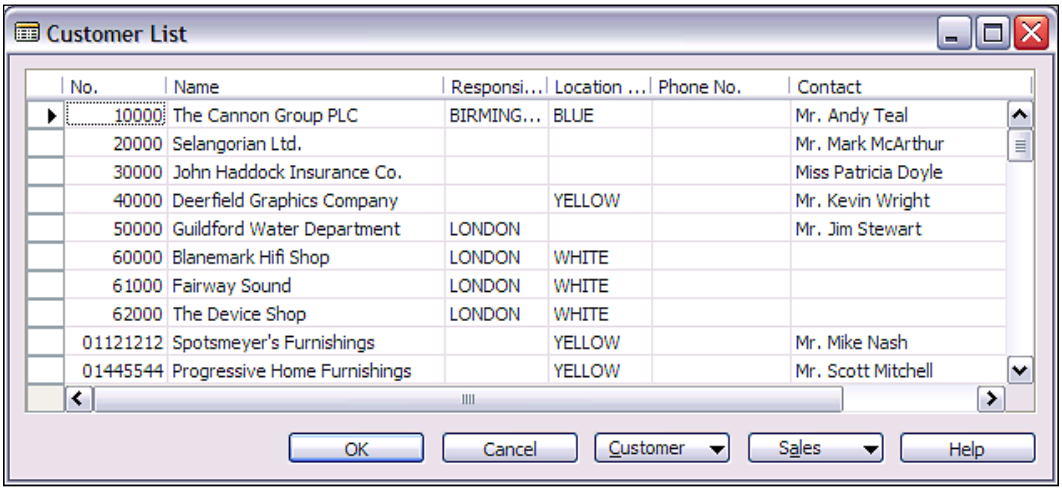
Card Forms

Card forms display one record at a time. These are generally used for the entry or display of `Master` table records. For example, Customer card for customer data, Item card for Inventory items, and G/L Account card for General Ledger accounts. Card forms often have multiple pages (tabs) with each tab on the **Customer Card** (for example) focusing on a different set of related customer data. Card forms for Master records display all the fields into which data must be entered by users. Typically, they also display summary data about related activity so that the Card form can be used as the primary inquiry form for its Master records. The following screenshot is a sample of a standard Customer card:



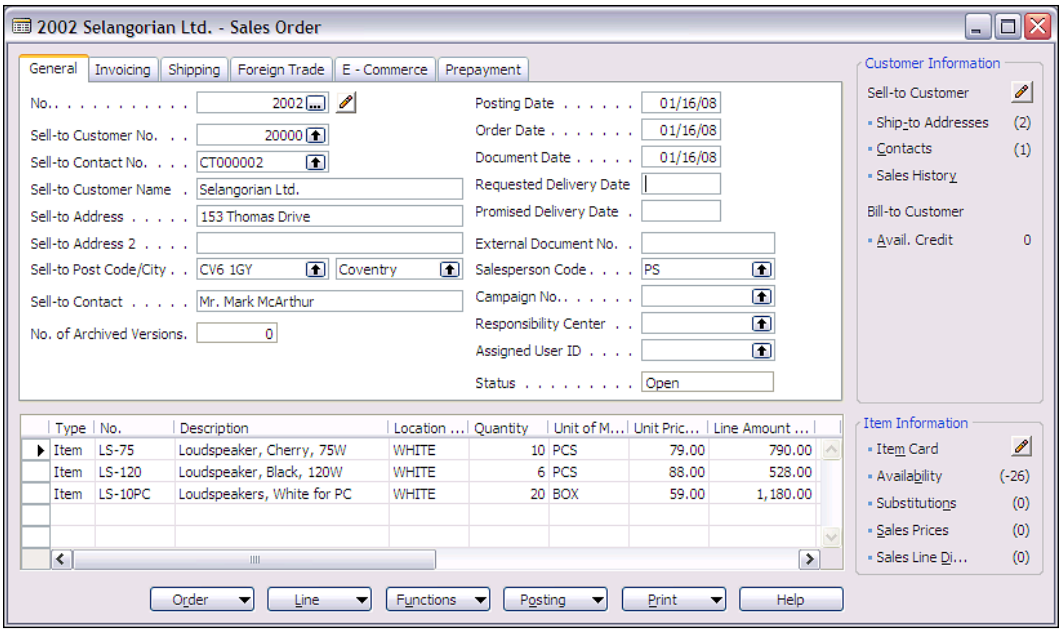
Tabular Forms

Tabular forms display a simple list of any number of records in a single table. The **Customer List** form in the following screenshot shows a subset of the data for each customer displayed. Master record list show fields intended to make it easy to find a specific entry. Tabular forms for list often do not allow entry or editing of the data. Tabular forms such as those for Journals are inherently intended for data entry.



Main/Sub Forms

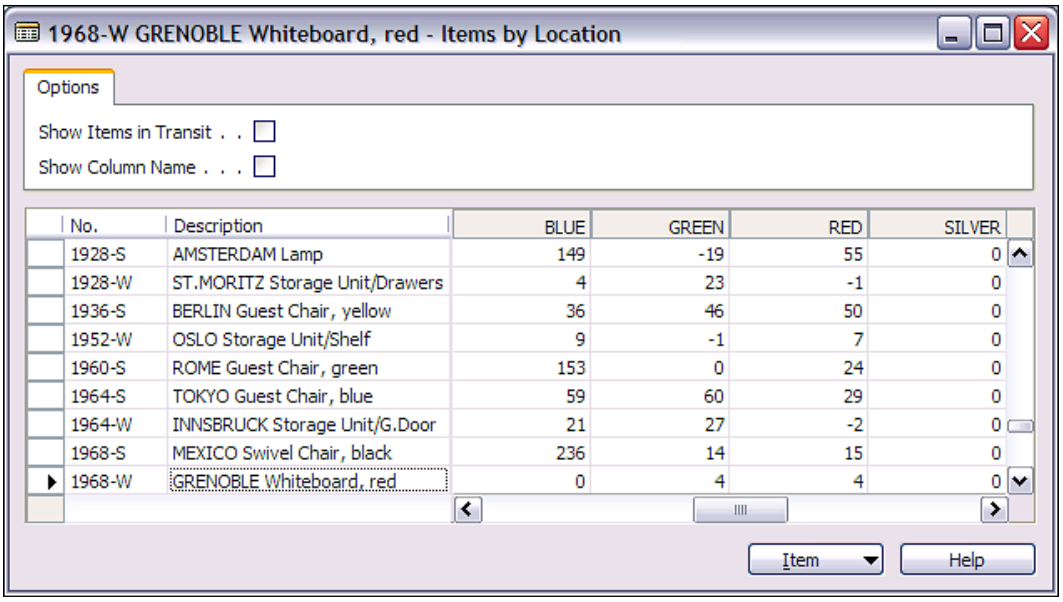
Another form style within NAV consists of a Card form plus a List form. These are called Main/Sub forms and are also referred to more casually as Header/Detail forms. An example is the **Sales Order** form as shown in the following screenshot. In this example, the upper portion of the form (the Main form) is a Card form with several tabs showing Sales Order data fields that have one occurrence. The lower portion of the form (the Subform) is a Tabular form showing a list of all the line items on the Sales Order. Line items may include product to be shipped, special charges, comments and other pertinent order details. The information to the right of the data entry is related data and computations which has been retrieved and formatted. On top of the form, the information is for the Ordering customer and the bottom contains information for the item on the selected line.



Matrix Forms

Matrix forms display multiple records at one time, and are also used to display the "intersect" of two related tables. For example: a spreadsheet-style matrix form showing the "intersect" (stock on hand) for each item at each location. The **Items** (**No.** and **Description**) shown on the Y axis (vertically) in the leftmost column and the **Locations** on the X axis (horizontally) across the top, and each intersect point contains the count of inventory for an item at a location.

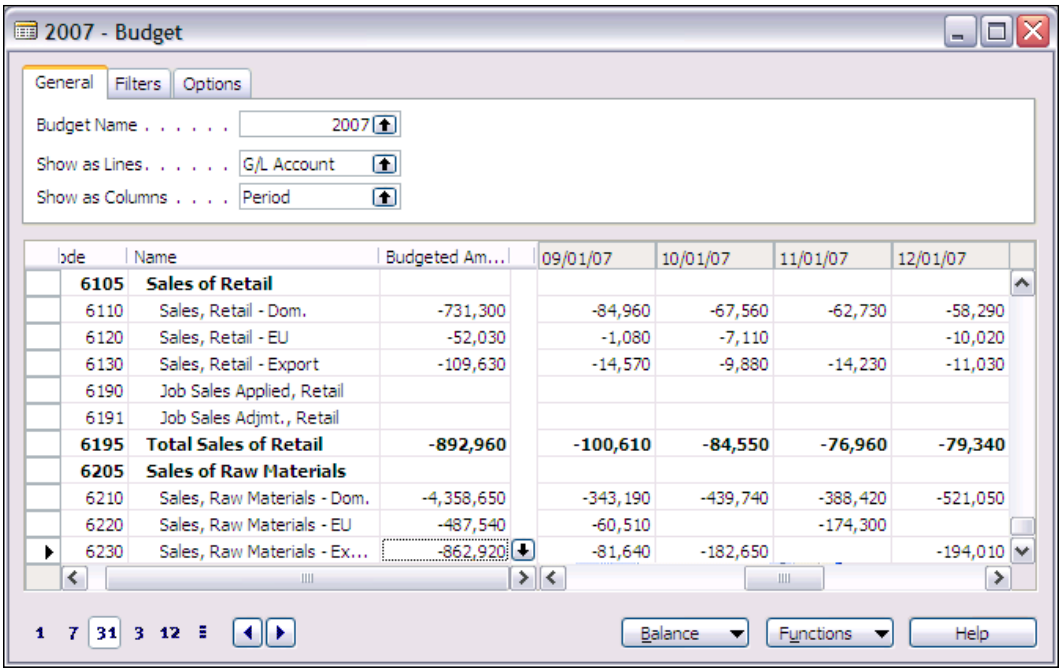
In the following screenshot, the **AMSTERDAM Lamp**, item number **1928-S**, has **149** lamps in stock in the **BLUE** warehouse and **55** in the **GREEN** warehouse. At the same time, we show a negative inventory in the **RED** warehouse, indicating that we have probably processed shipments for product, for which the receipts have not yet been posted.



Trendscape Forms

Trendscape forms have similar format to Matrix forms but with the addition of **Trendscape Option Buttons** and their underlying logic, the Trendscape forms are used to display data which is time dependent. The X-axis of a Trendscape matrix form is always date based, which generally uses the system Virtual Date table. The Trendscape option buttons allow the filtering and calculation of displayed information based on various accounting periods selected by the user. A Trendscape form is generally used for data that needs to be reviewed in summary form by various accounting periods (weeks, months, quarters, etc).

The sample Trendscape form in the following screenshot shows **Budget** data by date. This image is summarized on a monthly basis (i.e. the **31** button is selected). The top of this form, allows the entry for frequently used filter data, the left column (Y axis) shows the budget accounts, the X axis heading shows the date ranges for each column and the individual cells display the budgeted total for each row-column intersect (i.e. for each budget account by period). Specifically, the budget for **Total Sales of Retail** for the month period starting **09/01/07** is **-100,610**.



All Forms

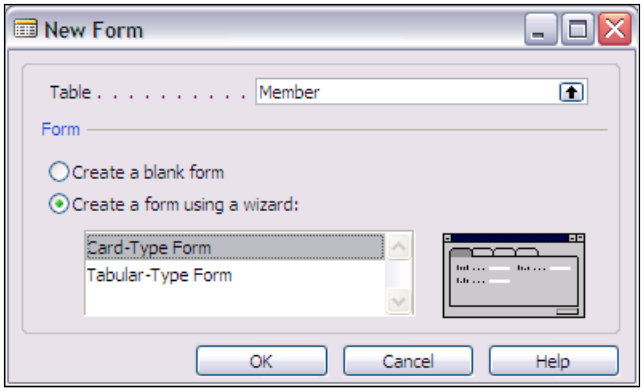
A Form consists of Form properties and Triggers, Controls, Control properties and Triggers. Data controls, generally are either labels displaying constant text or graphics, or containers that display data or other controls. Controls can also be elements such as buttons, menu items, and subforms. While there are a few instances where you must include C/AL code within form or form control triggers, in general it is a good practice to minimize the amount of code embedded within forms. Most of the time, any data related C/AL code can (and should) be located within the table object rather than the form object.

Creating a Card Form

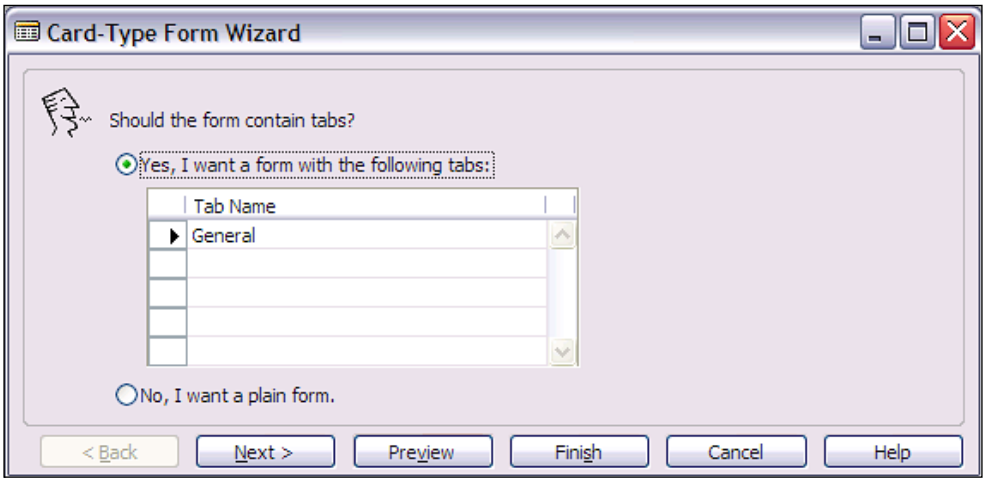
Let us try creating a Card form and a Tabular List form for the table we had created a little while ago. The NAV IDE consists of some object generation tools (i.e. the Wizards) to create basic forms and reports. These tools are useful either to create simple objects or as a starting place for more complicated objects. One of the (many) nice features of C/SIDE is that you can generate an object, then climb into the generated object and modify as though you had done all the coding from scratch by hand. This generation process is a one way process; once the generated object has been accepted and turned into an object under development, it cannot be manipulated within the form wizard again. For this reason, all of your wizard

design works for a particular form needs to be done at one time, before any manual object manipulation occurs. The Form Wizard and Form Designer tools are available to anyone who has a license containing the Form Designer granule.

We will be using the Form Wizard to create both of our forms. Later we will do more with these forms, but for now we will just see how the Form Wizard works. Open the **Object Designer**, click on **Form** and then click on **New**. The Form Wizard's first screen will appear. Enter the name (**Member**) or number (50000) of the **Table** with which you want the form to be associated (bound). Choose the option **Create a form using a wizard**. This time choose a **Card-Type Form**. Then click on **OK** as shown in the following screenshot:



The next screenshot provides the option of creating a plain form (no tabs) or a tabbed form with one or more tabs. We can also name the tabs on this screen. This time, even though it over skills at the moment to have a tabbed Card form, let us generate a form with one tab and use the default tab label of **General**. Now click on **Next**.



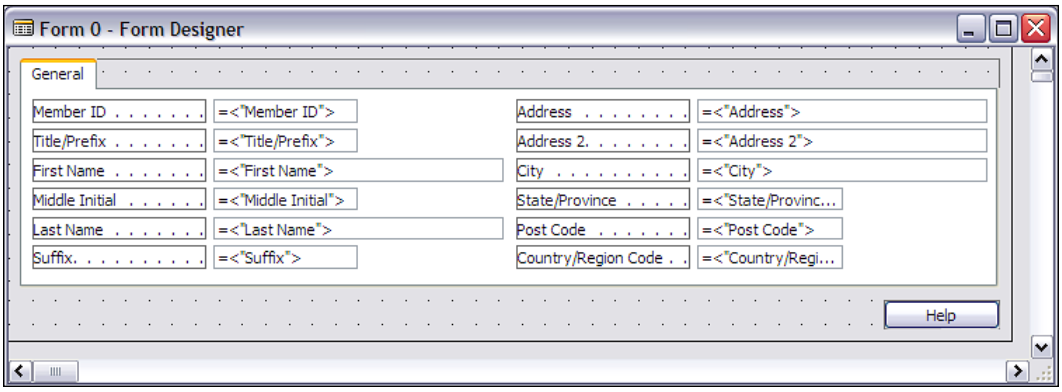
The next step is to choose what fields are to appear on our new form and how they will be placed. In this case, we are going to use all the fields and simply put them into two columns. First, let us take a quick walk through the controls on the Wizard form so we know what our basic toolkit is. There are six buttons associated with the design of the Card form on which we are working as shown in the following screenshot:

Button	Description
>	Choose one of the available fields
>>	Choose all the available fields
<	Remove one field from the object
<<	Remove all fields from the object
Separator	Insert a visual Separator (i.e. a horizontal line)
Column Break	Insert a Column Break (i.e. create a column split by ending one column and beginning another)
Preview	Display a Preview (screen display) of the object being created in the Wizard

The left column shown in the following screenshot, having heading as **Available Fields**, lists all the fields from the source table that have not yet been chosen for the Form object. The right column, headed as **Field Order**, lists all the fields that have been chosen in the order in which they will appear on the form.

In the following screenshot, you can see that we have split the **Field Order** column into name information and address information. It is not a very elegant design but right now we are still in first grade, i.e. just learning basics. We will deal with appearance issues later. Put all the fields in your table, on the form.

At any point during your work, you can take a sneak peek at what you are creating (i.e. click on the **Preview** button). When you are done with all the fields on your new form and you are satisfied with the layout, click on **Finish**. You will get the generated form object in the **Form Designer** as shown in the following screenshot:



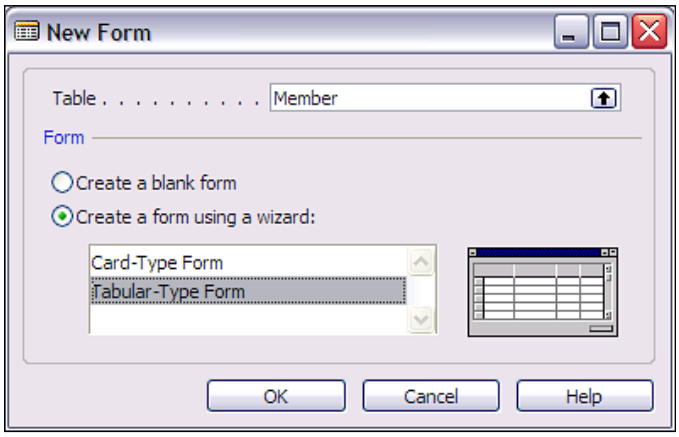
We are now through with the Form Wizard and have transitioned into the Form Designer. The Graphical User Interface Guidelines for NAV Card forms tells us that we should have all the long fields on the left side of the form and only short fields in the right column. Doing that with this form would have allowed us placing most, if not all, of the fields in the left column. Obviously, there is an opportunity here for some subjective form design discussion or debate (which we are going to leave unaddressed). At the moment, we are going to leave our form as it is. If you want to experiment with different layouts later, that would be a good way to practice using some of the C/SIDE tools.

If we want to modify the form manually, we could do that now. If yes, it would be a good idea to first save what we have done so far. We will do that the same as we did when we saved our table earlier. Press *Esc* and respond **Yes** to the **Do you want to save the changes to the Form** query.

Now enter the Form number (ID), you want to assign (50000) and name (Member Card). We are using the number 50000 just because that is the first custom form object number available to us with our Form Designer granule license. After, you have got the form saved so you don't lose what you've done so far, in another situation you could start making manual changes. But we're not ready for that level of difficulty yet. We will get into that in the Chapter covering Forms.

Creating a List Form

Our next task is to use the Form Wizard to create a tabular Member List form. Open the **Object Designer**, click on **Form** and then, click on **New**. Once again the Form Wizard's first screen will appear. Enter the name (Member) or number (50000) of the table with which you want the form to be associated. Choose the option **Create a form using a wizard**. This time choose the option to create a **Tabular-Type Form**. Then click on **OK** as shown in the following screenshot:



Now you will have the opportunity to choose which data fields will appear on each line of the tabular display. When List forms are designed for some type of referential lookup, generally they don't contain 100% of the data fields available, especially when working with a larger table. So let's choose a subset of the data fields to be displayed, just enough to make the display meaningful and easy to use.

Remember we can always return to the created form and easily add fields we left off or remove something we decide is not needed. In addition, NAV forms include a feature, which allows you to have some field columns identified as Not Visible by default. This property is field specific and controls whether the column for a data field displays on screen or not.

On a tabular form, even the non-programmer user can change the Visible property of each column to create a customized version. This user customization is tied to the individual user login and recorded in their `ZUP` file. `ZUP` files records user specific system state information so it can be retrieved when appropriate. In addition to user screen changes, the `ZUP` file records the identity of the most recent record in focus for each screen, the most recent contents of report selection criteria and request form field contents and a variety of other information. When the user returns to a screen or report, whether in the same session or after logout and return, data in the `ZUP` file helps restore the state of various user settings. This feature is very user friendly.

Let us choose just basic Name and Address fields for our initial List form as shown in the following screenshot.

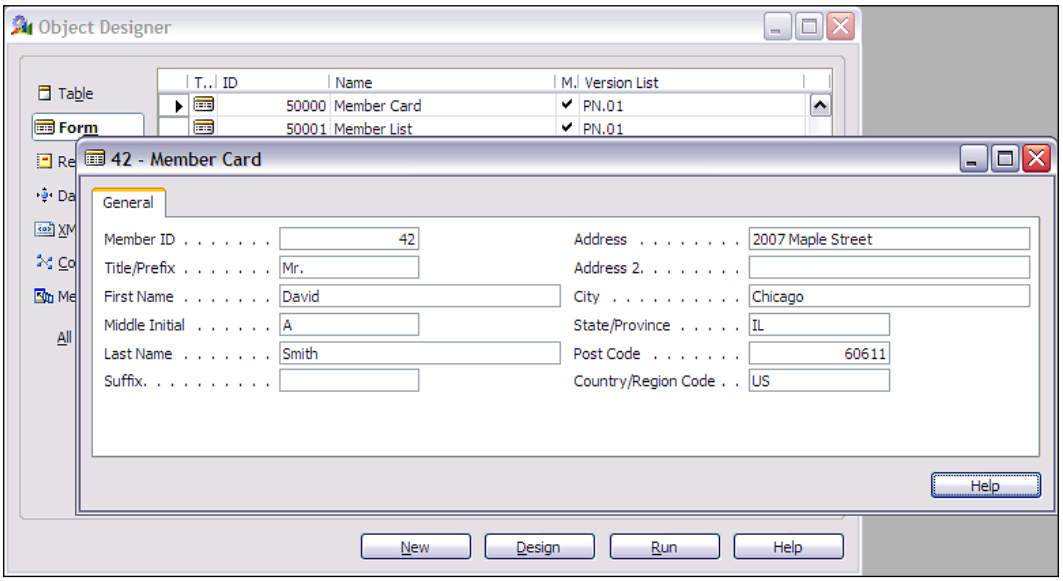
The Form Wizard functions are essentially the same for Tabular forms as it does for Card forms.

As with the Card form, at any point during your work you can take a sneak peek at what you are creating (i.e. click on the **Preview** button). If you feel like experimenting, you could move fields on and off the form or put fields in different orders. If you do experiment, use **Preview** to check the effects of your various actions. If your form gets hopelessly confused, that happens to all of us sometimes, just click on *Esc*, but be careful *not* to save the results and then start over. When you are done with all the fields on your new form and you are satisfied with the layout, click on **Finish**.

We are now done with the Form Wizard for our new List form and have transitioned into the Form Designer. If we want to modify the form manually, we could do that now. As before, just press *Esc* and respond **Yes** to the **Do you want to save the changes to the Form** query. Enter the Form number (ID), you want to assign (50001) and name (Member List). If you reuse the Form object number 50000, you would have overwritten the Card form, you created earlier.

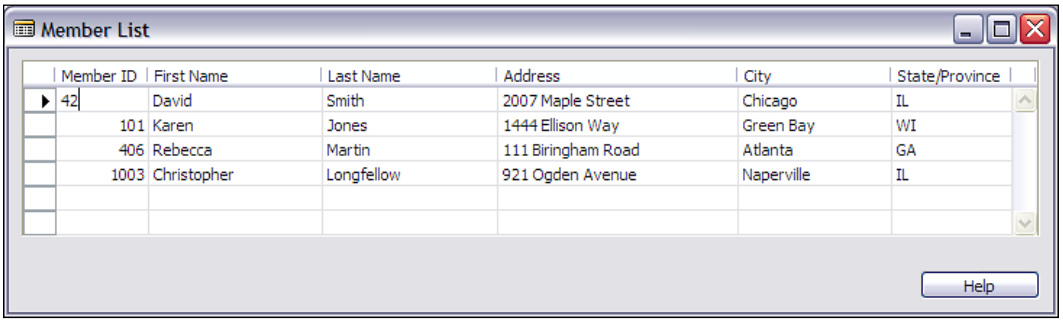
At this point, we have a data structure (Table 50000 – Member), a form to enter and maintain data (Form 50000 – Member Card) and a form to display or inquire into a list of data (Form 50001 – Member List). Let us use our Member Card to enter some data into our table. In a full application, we would be accessing our form from a Menu.

But for now, we will just run our form directly from the Object Designer.



Choose **Object Designer | Form**, highlight the line for form 50000 (as shown in the background of the preceding screenshot), and then click the **Run** button at the bottom of the Object Designer form. You should now see your Member Card on the screen similar to the preceding screenshot, but with all fields empty (blank). Enter data into the fields. Put in your name and address for the first entry (just for fun).

When you have finished making the entry, press *F3* (the NAV **New Record** key) to file away the data just entered and prepare it for the entry of a new record. Do that. Enter at least a couple more Member records as shown in the following screenshot both for your experience of doing the data entry and to generate several Member records to use for later testing.



Now we are going to look at the data through the List form you created. Exit the Member Card form, highlight Form 50001, Member List, and then **Run** it. You will see a list of all the entries you just entered, in order by the Member ID number.

Click on **Help** on the main menu above the main toolbar. Choose **Overview of F Keys** and study the information shown by pressing *Shift, Control* and *Shift* plus *Control*. These are the standard F Key functions throughout NAV.

Hopefully you are beginning to get a feeling for the ease of use and power of C/SIDE and NAV. Now that you have taken a quick look at where the data is defined and where it is entered, we will look at from where it is extracted and reported.

Reports

Report objects can be used for several purposes, sometimes more than one purpose at a time. One type is for the display of data. This can be on-screen (called Preview mode) or output to a printer device (Print mode). The displayed data may be as read from the database or may be the result of significant processing. As you would expect, a report may be extracting data from a single table (e.g. a list of customers) or a combination of several tables (e.g. a Sales analysis report).

Another type of report object processes data, without any display formatted output of the processed results. Typically, such reports are referred to as batch processing reports. Reports can also be used as containers for program logic to be called from other objects or executed as an intermediate step, in some sequence of processes. Normally this task would be handled by codeunit objects, but you could also use report objects, if you wanted to.

Report formats are limited to a combination of your imagination, your budget and the capabilities of NAV reporting. On one hand, NAV reporting is very powerful and flexible in terms of what data can be reported, plus the various types of filtering and combining can be done. On the other hand, NAV reporting is relatively limited in terms of formatting.

You can do most of what is normal and needed in terms of textual formatting, but have very limited graphical capability and almost no color capability. For these reasons, there are a number of good products, both from Microsoft and from third party vendors, to provide additional reporting capabilities complementary to those within NAV. New to V5.0 is a built-in functionality to use XML Style Sheets to support, exporting data to programs outside of NAV, especially those that are part of Microsoft Office such as Word and Excel.

Common report formats in NAV include document style (e.g. Invoices or Purchase Orders) also called as Form-Type, list style (e.g. Customer List, Inventory Item List, Aged Accounts Receivable) also called as Tabular-Type, and label format style (e.g. a page of name and address labels) called as Label-Type.

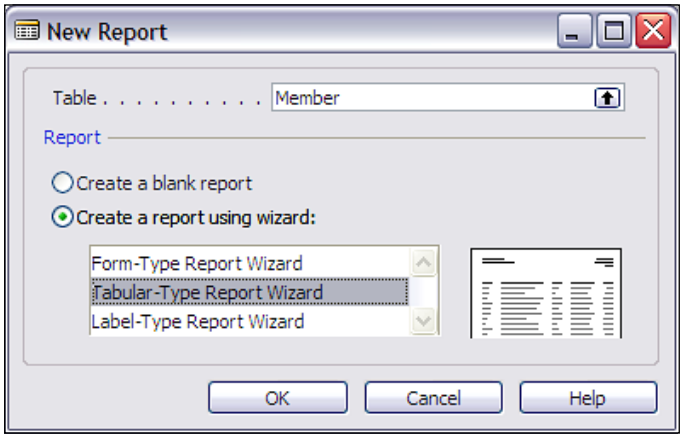
A significant aspect of the NAV report object is the built-in read-then-process looping structure which automates the sequence of read a record, then process it, and then read the next record. When manually creating or enhancing reports, you can define one data structure as subordinate to another, thus creating nested read-then-process loops. This type of predefined structure has its own good points and bad points. The good points usually relate to how easy it is, to do the kind of things it is designed for and the bad points relate to, how hard it is to do something that the structure doesn't anticipate. We will cover both sides of discussion when we cover Reports in more detail.

Creating a List Format Report

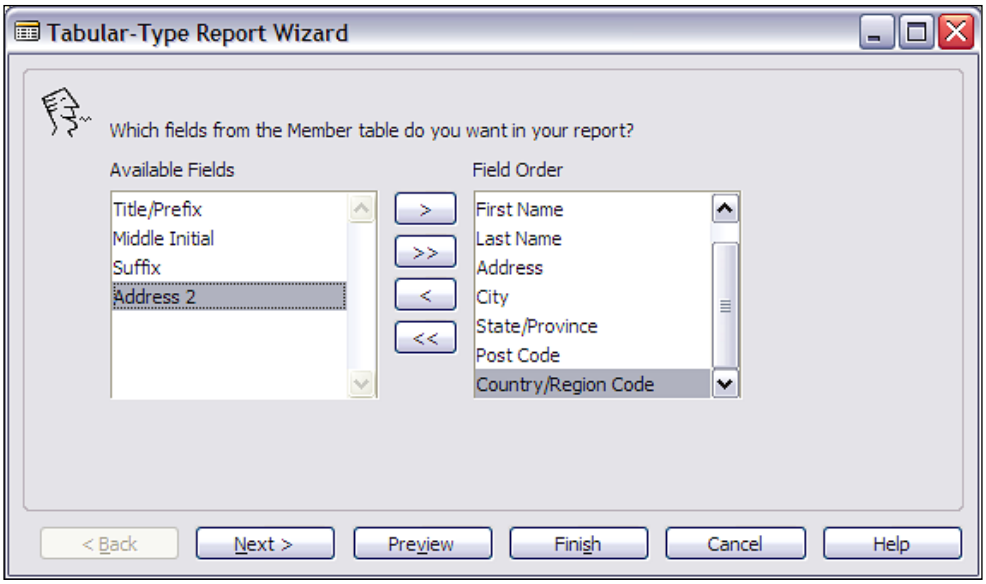
Let us create a simple list format report based on our Member table. We will use the Report Wizard this time. Just like the Form Wizard, the Report Wizard is quite useful for simple reports. The Report Wizard and Report Designer tools are available to anyone who has a license containing the Report Designer granule.

When you are doing more complex reports, it is often only modestly helpful to start with the Report Wizard. For one thing, the Report Wizard only deals with a single input table. However, even with complex reports, sometimes it is a good idea to create a little test version of some aspect of a larger report. Then you may create your full report without use of the Wizard, but letting the Report Wizard generate code to be your tutor in some aspect of layout or group totaling.

Open the **Object Designer**, click on **Report** and then, click on **New**. The Report Wizard's first screen will appear. Enter the name (Member) or number (50000) of the table with which you want the report to be associated. Choose the option **Create a report using a wizard**. This time choose a **Tabular-Type Report Wizard** to create a Member List. Then click on **OK** as shown in the following screenshot:



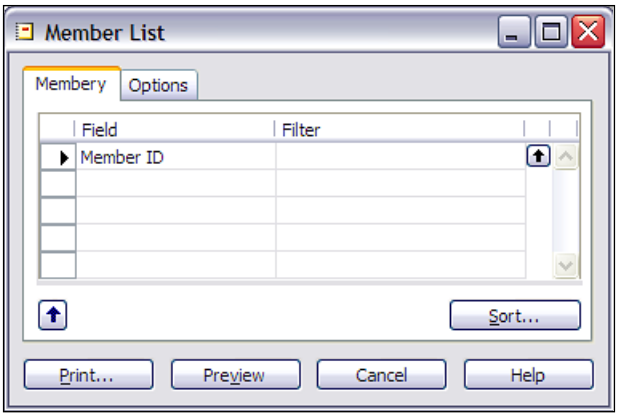
Next, you will be presented with a following screenshot for choosing, what data fields you want on your report. The order in which you place the fields in the **Field Order** column, top to bottom, will be the order in which they appear in your report, left to right.



Just as with the Form Wizard, you can **Preview** your report in the process of its creation, to see if you are getting the layout you want. More often with reports than with forms, you will perform quite a bit of manual formatting, after you finish with the Wizard. Because the report preview function utilizes the driver for the current default printer for formatting, make sure you have a default printer active and available, before you attempt to preview an NAV report.

After you have chosen the fields you want on your report (some are suggested in the preceding screenshot), click on **Next**. This will bring up a screen allowing you to predefine a processing/printing sequence for this report. You can select from any of the defined keys. At the moment our `Member` table only has one key so let's choose the **No, I don't want a special sorting of my data option**. As you will see later, that will generate a report where you can choose the sort sequence for each run. Click on **Next.**

Now you can choose a List Style or a Document Style layout. Click on each one of them to get an idea of the difference. If you like, **Preview** the report in process for each of these options chosen. Note that when you **Run** a report (and previewing it, is running it), the first thing you see is called the Report request screen. A Report request screen from the **Member List** report would look like the following screenshot:



The Report request screen is where the user enters any variable information, data filters, page, printer setups and desires sort order to control each report run. At the bottom right of the screen is the **Sort...** button, which allows the user to choose which predefined data key will apply to a particular report run. The user can also choose to **Print** the report (output to a physical device) or **Preview** it (output to the screen). To start with, just click on **Preview** to see your currently chosen layout.

Once you are satisfied with the layout, click on **Finish**. Again, just as with the Form Wizard, you will now be in the Report Designer, ready to make manual changes (isn't consistency great?). Exit the Report Designer by pressing the Close window icon, saying **Yes**, you do want to save the changes, and then saving your new report as ID 50000 and Name "Member List". If you then run your new report, using the **Run** button, it should look much like the following screenshot:

Member May 13, 2007

CRONUS International Ltd. Page 1

Member ID	Title / Prefix	First Name	Last Name	Address Line 1	City	State / Provinc	Postal Zone Code	Country
42	Mr.	David	Smith	2007 Maple Street	Chicago	IL	60611	US
101	Ms.	Karen	Jones	1444 Ellison Way	Green Bay	WI	54301	US
406	Ms.	Rebecca	Martin	111 Birmingham Road	Atlanta	GA	30301	US
1003	Mr.	Christopher	Longfellow	921 Ogden Avenue	Naperville	IL	60540	US

There is a lot more to learn about the design, creation, modification, and control of Reports.

For the moment, we are done with our introduction to development, but will continue with our introductory review of NAV's object types.

Codeunits

A codeunit is a container for "chunks" of C/AL code to be run "inline" or called from other objects. These "chunks" of code are properly called as Functions. Since we said earlier, you could use a Report object as a container for code segments (i.e. functions), then why do we need codeunits? One reason is that early in the life of C/SIDE, only codeunits could be used in this way. Somewhere along the line, the C/SIDE developers decided to relax that particular constraint, but from a system's design point of view, there are still very good reasons to use codeunits rather than other objects for function containers.

One important reason is that, the license specifically limits access to the C/AL code within codeunits differently than that within reports. The C/AL code within a report can be accessed with a "lower level" license than, what is required to access the C/AL code in a codeunit. If your customer has license rights to the Report Designer, they can access C/AL code within Report objects. A large percentage of installations have Report Designer license privileges. But they cannot access C/AL code within codeunit objects unless they have access to a more expensive license with Developer privileges (i.e. Application Builder or Solution Developer).

Another reason is that the codeunits are better suited structurally to contain only functions. Even though functions could be placed in other object types, the other objects types have superstructures which relate to their designed primary use for forms, reports, etc. Use of such an object primarily as a repository for functions, designed to be called from other objects creates code that is often more difficult to interpret, use and maintain.

Codeunits act only as a container, within, for the C/AL coded functions. They have no auxiliary functions, no method of user interaction, no predefined processing or predefined anything. If you are creating one or two functions that are closely related to the primary activity of a particular report, but are needed both from within and outside of the report, then, by all means, include the function in the report. But otherwise, use a Codeunit.

There are several codeunits delivered as part of the standard NAV product which are really function libraries. These codeunits consist totally of utility routines, generally organized on some functional basis (e.g. associated with Dimensions or with some aspect of Manufacturing or some aspect of Warehouse management). Some developers create their own libraries of favorite special functions and include such a "function library" codeunit in systems on which they can work.

MenuSuites

What is a MenuSuite? MenuSuites are the objects which are displayed as User Menus. They differ considerably from the other object types, we have discussed earlier. MenuSuites have a completely different structure, they are also maintained differently. In older versions of NAV, menus were constructed as versions of Form objects. With the release of Version 4.0, MenuSuites were offered as a way of providing a User Interface essentially similar to that found in the Outlook Navigation panel. MenuSuites are also maintainable in a limited way by the end user without any special license requirements. In addition, MenuSuites have the advantage of only showing the menu items for which the user has permissions to access.

MenuSuite entries do not have maintainable properties or contain triggers. With the advent of MenuSuites, NAV developers lost the ability to embed C/AL code within the menus. The only customizations that can be done with MenuSuites is to add, delete or edit menu entries. Later, we will discuss more about how to work with (and around) MenuSuite constraints.

Dataports

Dataports are specialized objects designed to export and import data between the NAV database (either implementation) and external text files. Dataports allow for a limited set of external data formats, generally focussed around what are commonly referred to as "comma delimited" files. Not, that they literally have to be delimited only with commas, but that is the category of file structure.

Dataports can contain C/AL logic which applies to either the importing or the exporting process (or both). The internal structure of a dataport object is somewhat similar to that of a report object combined with a table object. Dataports are driven by an internal read-then-process loop similar to that in reports. Dataports contain field definitions that relate to the specific data being processed.

Dataports are relatively simple and quite flexible tools for importing and exporting data. The data format structure can be designed into the dataport as well as logic for accommodating editing, validating, combining, filtering, etc. of the data as it is passed through the dataport. Dataports can be accessed directly from a menu entry, in the same fashion as forms and reports.

XMLports

At first glance, XMLports are for importing and exporting data, similar to the Dataports. But XMLports differ considerably in their operation, setup and intended usage. XMLport objects can only be used for XML formatted data. They must be "fired off" by some other routine (i.e. cannot be run directly through a menu entry). XML stands for eXtensible Markup Language. XML is a markup language much like HTML. XML was designed to describe data so that it would be easier to exchange data between dissimilar systems, for example, between your NAV ERP system and your accounting firm's financial analysis and tax preparation system.

XML is designed to be extensible, which means that you can create or extend the definition so long as you communicate the revised XML format to your correspondents. There is a standard set of syntax rules to which, XML formats must conform. XML is becoming more and more important because most software uses XML. For example, the new versions of Microsoft Office are quite XML "friendly".

Integration Tools

These integration tools are designed to allow direct input and output between NAV databases and external, non-NAV routines. But they do not allow access to C/AL based logic. The internal business rules or data validation rules that would normally be enforced by C/AL code or trigger actions or various properties, do not come into play when the data access is by means of one of the following integration tools. Therefore, you must be very careful in their use.

- N/ODBC: NAV provides the standard ODBC interface between external applications (such as word, excel, delphi, access, etc) and the NAV database. This is a separately licensed granule.
- C/OCX: It provides the ability to use OCXes to interface with the NAV database. This is also a separately licensed granule.
- C/FRONT: It provides the ability to access the NAV database directly from the code written in languages, other than C/AL. Earlier, this type of interface was primarily coded in C, but with V4.0 SP1, we now have the ability to interface from various .NET languages. In future versions, this capability is likely to expand considerably. This too is a separately licensed granule.
- Automation: It allows access to registered automation controller libraries within Windows from in-line C/AL code (e.g. can directly push data into a word document template or an excel spreadsheet template from C/AL). Automation controllers cannot be used to add graphical elements to NAV but they can contain graphical user interfaces that operate outside of NAV. But, when it is feasible to use an automation controller for interfacing externally, it is a simple and flexible way to expand the capabilities of your NAV system.

Backups and Documentation

As with any system where you can do development work, careful attention to documentation and backing up your work is very important. C/SIDE provides a variety of techniques for handling each of these tasks.

When you are working within the Object Designer, you can backup individual objects of any type or groups of objects by exporting them. These exported object files can be imported in mass or one object at a time to recover the original version of one or more objects. When objects are exported to text files, you can use a standard text editor to read or even change them. If, for example, you wanted to change all the instances of the field name **Customer** to **Patient**, you might export all the objects to text and make a mass "Find and Replace". You won't be surprised to find out that making such code changes in a text copy of an object is subject to a high probability of error, as you won't have all the safety features of the C/SIDE editor, keeping you from hurting yourself.

You can also use the NAV Backup function to create backup files containing just system objects or including data (i.e. a typical full system backup). A developer would typically use backup only as an easy way to get a complete snapshot of all the objects in a system. Backup files cannot be interrogated as to the detail of their contents, nor can selective restoration can be done. So, for incremental development backups, object exporting is the tool of choice.

Internal documentation (i.e. inside C/SIDE, not in external documents) of object changes can be done in three areas. First is the Object Version List, a field attached to every object, visible in the Object Designer screen. Whenever a change is made in an object, a notation should be added to the Version List.

In every object type except *MenuSuites*, there is a Documentation trigger at the top of the object. That is the recommended location for noting a relatively complete description of any changes that have been made to the object. Then, depending on the type of object and the nature of the specific changes, you should also consider annotating each change in the code, so it can be easily identified as a change by the next developer looking at this object.

In short, everything you have learned earlier about good backup practices and good documentation practices applies, when doing development in NAV C/SIDE. This is true whether the development is new work or modifications of existing logic.

Summary

In this Chapter, we have covered the basic definitions of terms related to NAV and C/SIDE. Then, we followed with the introduction of seven types of NAV objects (Tables, Forms, Reports, codeunits, MenuSuites, Dataports and XMLports).We also had an introduction to Form and Report Creation Wizards through review and hands-on use with the beginning of an NAV Developer Association application. Finally, we briefed the tools, that we use to integrate with external entities and also discussed how different types of backups and documentation are handled in C/SIDE. Now that we have covered the basics in general terms, let's dive into the detail of the primary object types. In the next Chapter, we will focus on Tables.