MODULE 10: CODEUNITS

Module Overview

Codeunits are similar to containers for code. Codeunits can hold functions that are used repeatedly throughout large application projects. This enables functions to be incorporated easily instead of rewriting the same lines of code. In addition, there are several standard codeunits that can be used to enrich application features, such as codeunit **400, SMTP Mail**.

Objectives

The objectives are:

- Explain the concepts of codeunits.
- Provide an overview of designing codeunits.
- Provide an overview by using codeunits.
- Define variables and functions in a codeunit.
- Use the **SMTP Mail** codeunit.

Codeunit Fundamentals

C/AL code can be stored in a table, page, report, or other object. In a simple application, the common approach is to put the code in the object that calls the functions. However, as an application grows, developers frequently find that the same functions are used repeatedly throughout the application.

Instead of declaring the same functions repeatedly, you can define them one time. This is where the Microsoft Dynamics NAV Development Environment codeunits become useful. Think of a codeunit as a container for C/AL code to use in many application objects. In codeunits, developers can define functions, global variables, and temporary tables.

Functions

A function is a sequence of C/AL statements that developers define to create new functionality. Each function that is created in a codeunit is displayed as a separate trigger, known as a *function trigger*. A function trigger it has its own set of statements. Think of a codeunit as a container for multiple functions.

Local Variables

Within each function, developers can define local variables. A local variable's scope is limited to the function in which it is defined.

Global Variables

A global variable's scope covers all functions in the codeunit.

Temporary Tables

A temporary table is actually a record variable which has the Temporary property set to Yes. It resembles an actual table; however, it is not stored in the database. Temporary tables are mainly used as structured variables that hold data temporarily while developers work on it.

Triggers

All codeunits include two default triggers known as Documentation and OnRun. In the Documentation trigger, developers can add optional information about the code such as the purpose of the codeunit, a version number, and more. In the OnRun trigger, developers can include code that they want the system to execute when the codeunit is run.

***Note:** A codeunit can be run by using <CodeunitVariableName>.RUN. When a codeunit is run, the code that is written in the OnRun trigger executes.*

Design Codeunits

Codeunits are designed in the **C/AL Editor**, which is accessed directly from the Object Designer.

Use the C/AL Editor

The **C/AL Editor** is where code is viewed and changed. The following keyboard shortcuts can be used when you work in the **C/AL Editor**.

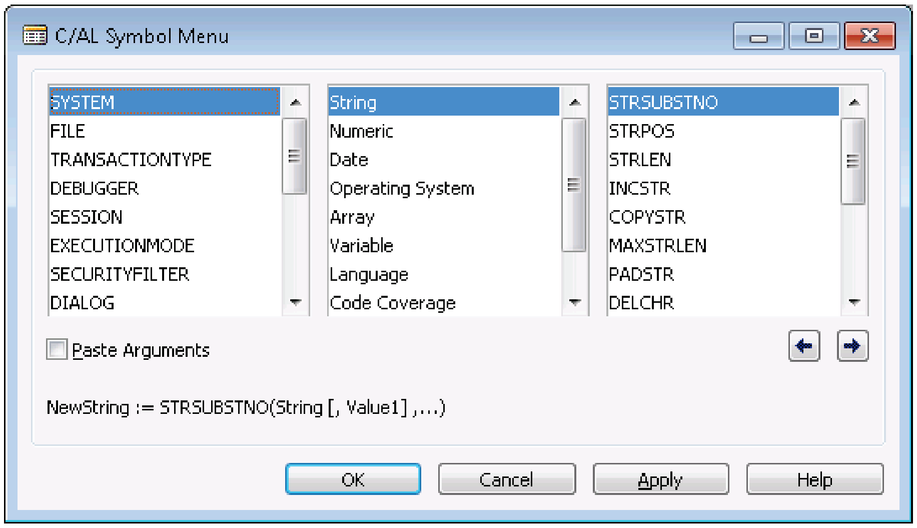
Keyboard shortcut	Description
CTRL+X	Deletes the selected text and places it on the clipboard.
CTRL+C	Copies the selected text to the clipboard.
CTRL+V	Pastes the text from the clipboard into the codeunit at the pointer position.
CTRL+F	Opens the **Find** window and searches for text.

Use the C/AL Symbol Menu

Access the **C/AL Symbol Menu** from the **C/AL Editor**. Use the **C/AL Symbol Menu** for an overview of:

- All variables that are defined in the codeunit
- All C/AL functions

Open **the C/AL Symbol Menu** by clicking **View**, **C/AL Symbol Menu** or by pressing F9. See the "C/AL Symbol Menu" figure.



```
NewString := STRSUBSTNO(String [, Value1] ,...)
```

FIGURE 10.1: THE C/AL SYMBOL MENU

The information in the **C/AL Symbol Menu** is divided into columns. The column on the left shows all defined variable names and function categories. The second column shows the subcategories, and the third column, on the right side shows the functions in the selected category.

View the syntax in the bottom of the window, and then click **OK** or **Apply**. The Development Environment inserts this string in the **C/AL Editor**. Clicking **OK** closes the **C/AL Symbol Menu** window automatically, but clicking **Apply** leaves it open.

In some cases, such as when a control on a page is a subpage or when a field is a **BLOB** field, there are more than three columns. Use the right arrow button to scroll the **C/AL Symbol Menu** columns to the right, and the left arrow button to scroll the columns to the left.

Use Codeunits

By using codeunits, developers eliminate the need to duplicate code and make the code easier to maintain. If you use the same code repeatedly in pages or reports, create a function in a codeunit and access it by using the following syntax.

Code Example

```
<CodeunitVariableName>.<FunctionName>
```

Run Codeunits

Codeunits contain the OnRun trigger. This executes when you run the codeunit. Run a codeunit in any of the following ways:

- Directly from the **Object Designer**, by selecting a codeunit, and clicking **Run**
- From a page action, by defining the RunObject property
- From code, by using the CODEUNIT.RUN statement
- From code, by using the RUN function on a codeunit variable

Any of these methods run the OnRun trigger in the referenced codeunit.

The following syntax shows how to run a codeunit from C/AL code without using a variable.

CODEUNIT.RUN example

```
[{Ok} := CODEUNIT.RUN(Number [, Record])
```

Access Codeunits through a Variable

When you run a codeunit through the CODEUNIT.RUN statement, you can only run its OnRun trigger. If a codeunit contains other functions, you must declare a variable to access those functions.

After you declare a variable of type Codeunit, you can access its functions by writing the name of the variable that represents the codeunit followed by a dot. Follow this with the name of the function.

Note: *This concept applies not only to codeunits, but to any application objects that can contain global C/AL functions.*

Codeunits may contain internal variables that are defined as global variables. These variables cannot be accessed directly from code outside the codeunit. However, you can access them through user-defined functions on the codeunit.

When you use a codeunit variable for the first time, a new instance of the codeunit is created. This means that a new set of internal variables is initialized so that different codeunit variables use different sets of internal variables.

Codeunit Variable Assignment

A codeunit variable does not contain a codeunit, but only a reference to a codeunit. More than one codeunit variable can refer to the same codeunit, as shown in the "Relation Between Codeunit Variables and the Codeunit" figure.

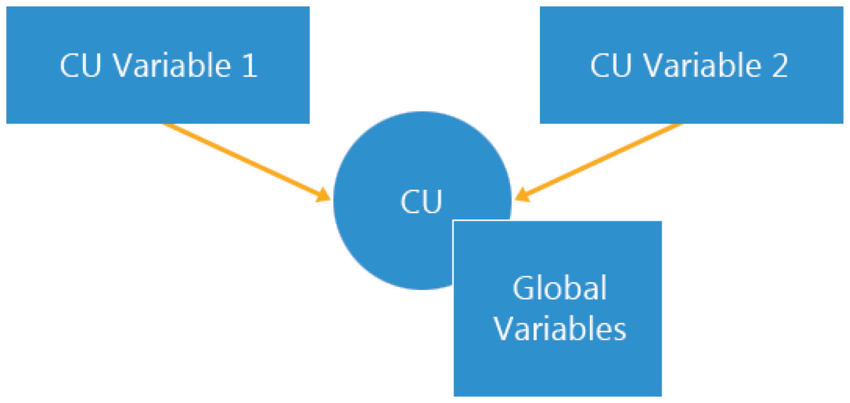


FIGURE 10.2: THE RELATION BETWEEN CODEUNIT VARIABLES AND THE CODEUNIT

Codeunits can be treated as objects. You can assign one codeunit variable to another by using an assignment statement. This creates a new reference to the same codeunit instance. In other words, the codeunit variables then use the same set of internal variables.

CLEAR on Codeunits

Use the function CLEAR on a codeunit variable that has a reference to a codeunit instance with two or more references. In these instances, the codeunit variable deletes the reference to the codeunit and not the actual codeunit instance. In other words, the codeunit stays intact and is available to other codeunit variables that may be assigned a reference to this codeunit.

To delete a codeunit instance, clear all references to the codeunit with the **CLEAR** function. To clear the internal variables in a codeunit, use the **CLEARALL** function from a user-defined function within the codeunit. When a local codeunit variable goes out of scope, it means that it is no longer used by the codeunit. It is automatically cleared.

Single Instance Codeunits

In some cases, only one instance of a codeunit must exist. This means that all the codeunit variables of a particular codeunit use the same set of variables. When the SingleInstance property of the codeunit is set to Yes, all the codeunit variables for that codeunit use the same instance. This lets developers create global variables.

Note: *Avoid using global variables for most code. However, in certain situations, you may have to use them. For example, use a global variable to make sure that only one instance of an external variable is used.*

A single instance codeunit is instantiated when it is used for the first time. Other codeunit instances (codeunits that do not have the SingleInstance property set to Yes) are deleted when the last codeunit variable that uses that codeunit instance goes out of scope. However, single instance codeunits remain instantiated until the current company is closed from the Microsoft Dynamics NAV client.

Codeunits Limitations

Global variables and temporary tables in a codeunit cannot be accessed directly from other application objects. The only way to access these values is through user-defined functions, which are created in the codeunit.

All C/AL functions can be used in a codeunit, although a function name cannot be the same name as a built-in function. Two or more user-defined functions cannot have the same name, unless they are part of different application objects.

Demonstration: Define and Use a Codeunit from C/AL Code

The following demonstration shows how to do perform the following tasks:

- Create a codeunit.
- Add global variables and functions in the **C/AL Globals** window.
- Use the codeunit both directly and indirectly from a variable.
- Assign a codeunit variable.

Demonstration Steps

1. Create a new codeunit and save it as Codeunit 90006, **Simple Codeunit**.
   a. In the **Object Designer**, click **Codeunit** and then click **New**.
   b. In the **C/AL Editor** window, click **File** > **Save**.

   c. In the **Save As** dialog window, in the **ID** field, enter “90006 ”.
   d. In the **Name** field, enter “Simple Codeunit”.
   e. Make sure that **Compiled** is selected, and then click **OK**.

2. Add a global variable named Name of type Text[30].
   a. Click **View** > **C/AL Globals**.
   b. On the Variables tab, define the following variables.

Name	DataType	Length
Name	Text	30

3. Add global functions named **SetName** and **GetName**. **SetName** receives a parameter of type Text[30], and **GetName** returns a value of type Text[30].
   a. On the **Functions** tab, define the **SetName** and **GetName** functions.
   b. Select the SetName function, and then click **Locals**.
   c. In the **SetName – C/AL Locals** window, on the Parameters tab, create the following parameter.

Name	DataType	Length
NewName	Text	30

   d. Close the SetName – C/AL Locals window.
   e. On the **Functions** tab of the **C/AL Globals** window, select the GetName function, and then click **Locals**.
   f. In the **GetName – C/AL Locals** window, on the Return Value tab, in the **Return Type** field, type “Text”.
   g. In the Length field, type “30”.
   h. Close the GetName – C/AL Locals window.
   i. Close the **C/AL Globals** window.

4. Add code to the **SetName** function to set the value of the *Name* variable to the value of the *NewName* parameter.
   a. In the **SetName** function trigger, enter the following C/AL code.

SetName Trigger

```
Name := NewName;
```

5. Add code to the **GetName** function to return the current value of the *Name* variable.
   a. In the **GetName** function trigger, type the following C/AL code.

GetName Trigger

```
EXIT(Name);
```

6. Compile, save, and then close the codeunit.
   a. Click **File > Save**.
   b. In the **Save** dialog box, make sure that **Compiled** is selected, and then click **OK**.
   c. Close the **C/AL Editor** window.

7. Create another codeunit, and save it as Codeunit 90007, **Simple Assignment**.
   a. In the **Object Designer**, click **Codeunit** and then click **New**.
   b. In the **C/AL Editor** window, click **File** > **Save**.
   c. In the **Save As** dialog box, in the **ID** field, type "90007".
   d. In the **Name** field, type "Simple Assignment." make sure that **Compiled** is selected, and then click **OK**.

8. Define two global variables, *Cu1*, and *Cu2*, of type Codeunit 90007.
   a. Click **View** > **C/AL Globals**.
   b. On the **Variables** tab, define the following variables.

Name	DataType	Subtype
Cu1	Codeunit	Simple Codeunit
Cu2	Codeunit	Simple Codeunit

9. Write the code in the OnRun trigger that performs the following tasks:
   - Calls the SetName function on the Cu1 variable.
   - Shows the result of the GetName function on both variables.
   - Assigns the Cu1 variable to Cu2 variable.
   - Shows the result again of the GetName function on both variables.

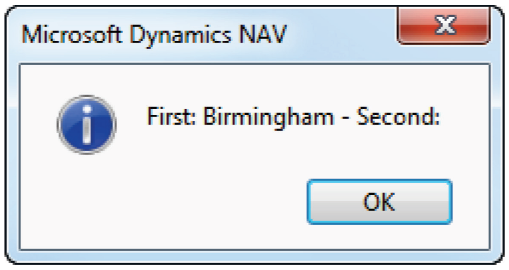
   a. In the OnRun trigger, enter the following C/AL code.

OnRun Trigger

```
Cu1.SetName('Birmingham');

MESSAGE('First: %1 - Second: %2',

  Cu1.GetName,

  Cu2.GetName);

Cu2 := Cu1;

MESSAGE('First: %1 - Second: %2',

  Cu1.GetName,

  Cu2.GetName);
```

10. Compile, save, and then run the codeunit.
    a. Click **File** > **Save**.
    b. In the **Save** dialog box, make sure that **Compiled** is selected, and then click **OK**.
    c. Close the **C/AL Editor** window.
    d. In **Object Designer**, locate the Codeunit 90007, **Simple Assignment**, and then click **Run**.

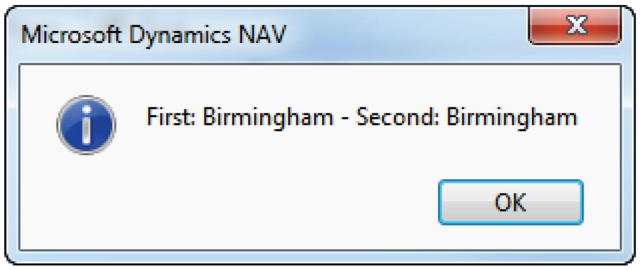
The first message box shows only the value of the *Name* variable in the first codeunit variable, because that is where the variable assignment occurred.


Microsoft Dynamics NAV

First: Birmingham - Second:

FIGURE 10.3: FIRST MESSAGE

After *Cu1* was assigned to *Cu2*, both variables refer to the same instance of the codeunit in memory. Therefore, the message box now shows different results.



First: Birmingham - Second: Birmingham

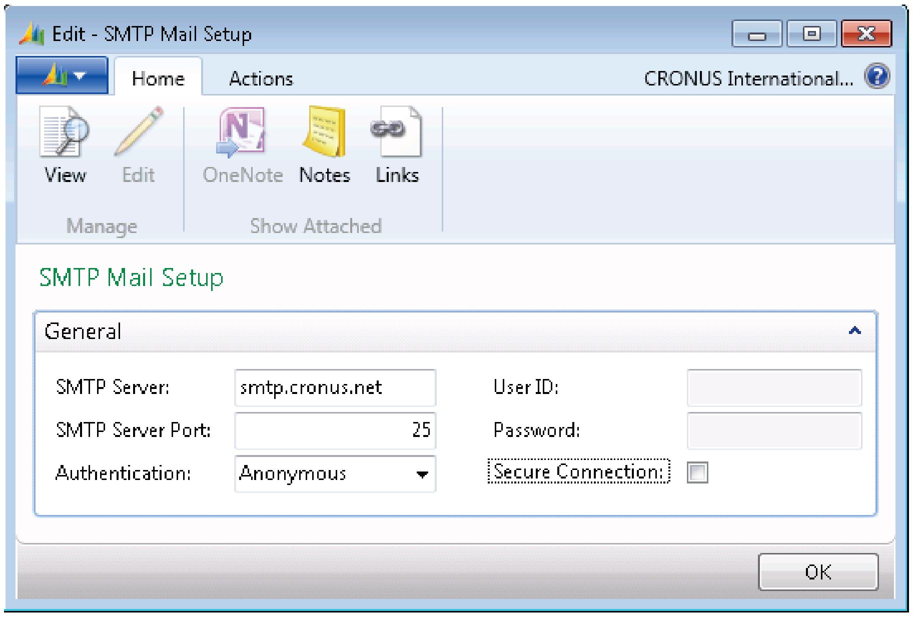
FIGURE 10.4: MESSAGE AFTER ASSIGNMENT

SMTP

Microsoft Dynamics NAV 2013 has the ability to send email by using Simple Mail Transfer Protocol (SMTP). Unlike sending email programmatically by using the Mail Application Programming Interface (MAPI), sending email by using the SMTP functionality in Microsoft Dynamics NAV does not require an email client, or detailed knowledge of the SMTP protocol.

SMTP Mail Setup

Before you use the SMTP functionality, you must provide the SMTP server and account information by using the **SMTP Mail Setup** window (page 409). See "SMTP Mail Setup Page".



SMTP Server:	smtp.cronus.net	User ID:	
SMTP Server Port:	25	Password:	
Authentication:	Anonymous	Secure Connection:	☐

FIGURE 10.5: THE SMTP MAIL SETUP PAGE

With the SMTP server information active, you can use the functions that are defined in the SMTP Mail codeunit (codeunit 400).

SMTP Mail Codeunit

The **SMTP Mail** codeunit defines the following functions:

- CreateMessage
- TrySend
- Send
- AddRecipients
- AddCC
- AddBCC
- AppendBody
- AddAttachment
- AddAttachmentStream
- CheckValidEmailAddresses
- CheckValidEmailAddress
- GetLastSendMailErrorText
- GetSMTPMessage

For simple email, only the functions **CreateMessage** and **Send** are necessary. Use the **CreateMessage** function to configure a message. The **Send** function performs the actual sending. For example, assuming that codeunit 400 is declared as a global variable named *SMTPCU*, the following code sends an email message.

Code Example

```
SMTPCU.CreateMessage(
       'CRONUS International Ltd.',
       'support@cronuscorp.net' (mailto:'support@cronuscorp.net'),
       'oakvilleworld@cronuscorp.net' (mailto:'oakvilleworld@cronuscorp.net'),
        'Issue Status',
       'Your issue has been resolved',
       FALSE);
 SMTPCU.Send;
```

You can use several functions after the **CreateMessage** function and before the **Send** function, to append additional information to the message. These functions are as follows.

Function	Description
TrySend	Use this function to send a message, based on the parameters set in the **SMTP Mail**

Function	Description
	Setup Page.
AddRecipients	Use this function if there are more receiver email addresses than can fit into the 1024 characters that are permitted for the Recipients parameter in the CreateMessage function.
AddCC	Use this function to add carbon copy recipients.
AddBCC	Use this function to add blind carbon copy recipients.
AppendBody	Use this function if the message body is longer than the 1024 characters permitted in the Body parameter in the CreateMessage function.
AddAttachment	Use this function to provide the path of the file to attach to the email message.
AddAttachmentStream	Use this function to pass an **Instream** to attach to the email message.

Use the functions **CheckValidEmailAddresses** and **CheckValidEmailAddress** to check whether a text string is for a valid email address, or valid multiple email addresses.

Use the last two functions to return possible error texts or messages that are generated by the SMTP-server when you send an email.

Module Review

Module Review and Takeaways

Codeunits are application objects that primarily serve as containers for code to aid development and code management. You create and change them in the **C/AL Editor**. Unlike other objects, a codeunit can be a single instance object. Every variable that refers to it refers to the same instance of the codeunit. There are several standard codeunits that you can use to enrich application features, such as codeunit 400, **SMTP Mail**.

Test Your Knowledge

Test your knowledge with the following questions.

1. What triggers are available in every codeunit?

2. What key do you press to open the **C/AL Symbol Menu**?

3. How do you call a function in a codeunit from another object?

4. Can two codeunit variables refer to the same instance of a codeunit?

5. Can two codeunit variables that are declared in the same object refer to different instances of a codeunit?

6. What change has to be made to a codeunit variable to point to the same instance of a codeunit as another codeunit variable?

7. What function can you use to make a codeunit variable point to a new instance of a codeunit?

8. If a codeunit is designed as a single instance codeunit, can two codeunit variables refer to different instances of that codeunit?

9. What is the correct way to address a global variable of a codeunit from an object that has a variable that is named *CU* for that codeunit?

10. Can you create a function named Power in a codeunit?

Test Your Knowledge Solutions

Module Review and Takeaways

1. What triggers are available in every codeunit?

   MODEL ANSWER:

   Documentation and OnRun

2. What key do you press to open the **C/AL Symbol Menu**?

   MODEL ANSWER:

   F5

3. How do you call a function in a codeunit from another object?

   MODEL ANSWER:

   Declare a variable of type Codeunit. Then, call the function using the Codeunit.Function syntax.

4. Can two codeunit variables refer to the same instance of a codeunit?

   MODEL ANSWER:

   Yes.

5. Can two codeunit variables that are declared in the same object refer to different instances of a codeunit?

   MODEL ANSWER:

   Yes.

6. What change has to be made to a codeunit variable to point to the same instance of a codeunit as another codeunit variable?

   MODEL ANSWER:

   You assign one codeunit variable to another.

7. What function can you use to make a codeunit variable point to a new instance of a codeunit?

   MODEL ANSWER:

   CLEAR

8. If a codeunit is designed as a single instance codeunit, can two codeunit variables refer to different instances of that codeunit?

   MODEL ANSWER:

   No.

9. What is the correct way to address a global variable of a codeunit from an object that has a variable that is named *CU* for that codeunit?

   MODEL ANSWER:

   You cannot access global variables of a codeunit from other objects. You only can access functions of the codeunit. Also, you only can access those functions that are not marked as Local.

10. Can you create a function named Power in a codeunit?

    MODEL ANSWER:

    Yes.