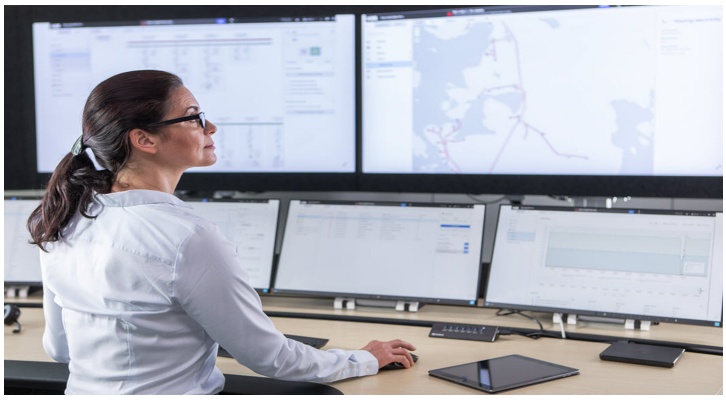
GRID AUTOMATION PRODUCTS

MicroSCADA X SYS600 10.2

Visual SCIL Application Design



Document ID: 1MRK 511 483-UEN
Issued: March 2021
Revision: A
Product version: 10.2

Table of contents

Section 1 Copyrights

The information in this document is subject to change without notice and should not be construed as a commitment by Hitachi Power Grids. Hitachi Power Grids assumes no responsibility for any errors that may appear in this document.

In no event shall Hitachi Power Grids be liable for direct, indirect, special, incidental or consequential damages of any nature or kind arising from the use of this document, nor shall Hitachi Power Grids be liable for incidental or consequential damages arising from the use of any software or hardware described in this document.

This document and parts thereof must not be reproduced or copied without written permission from Hitachi Power Grids, and the contents thereof must not be imparted to a third party nor used for any unauthorized purpose.

The software or hardware described in this document is furnished under a license and may be used, copied, or disclosed only in accordance with the terms of such license.

Trademarks

ABB is a registered trademark of ABB Asea Brown Boveri Ltd. Manufactured by/for a Hitachi Power Grids company. All other brand or product names mentioned in this document may be trademarks or registered trademarks of their respective holders.

Guarantee

Please inquire about the terms of guarantee from your nearest Hitachi Power Grids representative.

Third Party Copyright Notices

List of Third Party Copyright notices are documented in "3rd party licenses.txt" and other locations mentioned in the file in SYS600 and DMS600 installation packages.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (https://www.openssl.org/). This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Section 2 Introduction

2.1 Use of symbols

This publication includes warning, caution and information symbols where appropriate to point out safety-related or other important information. It also includes tips to point out useful hints to the reader. The corresponding symbols should be interpreted as follows:

Warning icon indicates the presence of a hazard which could result in personal injury.

Caution icon indicates important information or a warning related to the concept discussed in the text. It might indicate the presence of a hazard, which could result in corruption of software or damage to equipment/property.

Information icon alerts the reader to relevant factors and conditions.

Tip icon indicates advice on, for example, how to design a project or how to use a certain function.

Although warning hazards are related to personal injury, and caution hazards are associated with equipment or property damage, it should be understood that operation of damaged equipment could, under certain operational conditions, result in degraded process performance leading to personal injury or death. Therefore, comply fully with all warnings and caution notices.

2.2 Intended audience

This manual is intended for installation personnel, administrators and skilled operators to support installation of the software.

2.3 Document conventions

The following conventions are used for the presentation of material:

- The words in names of screen elements (for example, the title in the title bar of a dialog, the label for a field of a dialog box) are initially capitalized.
- Capital letters are used for file names.
- Capital letters are used for the name of a keyboard key if it is labeled on the keyboard. For example, press the CTRL key. Although the Enter and Shift keys are not labeled they are written in capital letters, e.g. press ENTER.
- Lowercase letters are used for the name of a keyboard key that is not labeled on the keyboard. For example, the space bar, comma key and so on.
- Press CTRL+C indicates that the user must hold down the CTRL key while pressing the C key (in this case, to copy a selected object).
- Press ALT E C indicates that the user presses and releases each key in sequence (in this case, to copy a selected object).
- The names of push and toggle buttons are boldfaced. For example, click **OK**.

- The names of menus and menu items are boldfaced. For example, the **File** menu.
  - The following convention is used for menu operations: **Menu Name/Menu Item/ Cascaded Menu Item**. For example: select **File/Open/New Project**.
  - The **Start** menu name always refers to the **Start** menu on the Windows Task Bar.
- System prompts/messages and user responses/input are shown in the Courier font. For example, if the user enters a value that is out of range, the following message is displayed: `Entered value is not valid`.
  The user may be told to enter the string MIF349 in a field. The string is shown as follows in the procedure: `MIF349`
- Variables are shown using lowercase letters: sequence name

2.4 Document revisions

Revision	Version number	Date	History
A	10.2	31.03.2021	New document for SYS600 10.2

Section 3 Visual SCIL overview

3.1 About this Section

This section introduces the basic concepts of the SYS600 user interface and summarizes the Visual SCIL user interface design method. It is organized into two sections with the following contents:

Section 3.2	This section discusses the basic concepts of the SYS600 user interface: pictures, dialogs and dialog items. It introduces the user interface design methods and provides recommendations regarding the use of pictures and dialogs. It also describes the possible views in different types of monitors.
Section 3.3	This section introduces the Visual SCIL objects, the dialogs, dialog items and dialog systems. The most common procedure for designing dialog systems is introduced as well as handling of Visual SCIL objects and an example.

3.2 SYS600 User Interface Design

3.2.1 User Interface Components

A SYS600 user interface is composed of one or more windows. Each window may provide illustrations of the controlled process, presentations of the alarm and event state, history of the process, graphical illustrations, operator dialogs, images and so on. SYS600 supports two different main window types, also known as monitors: Classic Monitors and Monitor Pro.

Classic Monitors are mainly supported for backwards compatibility reasons and this monitor type is fully compatible with SYS 500 Monitors. Monitor Pro represents the user interface generation introduced in SYS600.

There are three main user interface component types available:

- Process Displays
- Visual SCIL dialogs
- Pictures

Process Displays are introduced in SYS600 and are mainly used to illustrate the process. The Process Displays are built with Display Builder and they support for example zooming, panning and decluttering features.

Visual SCIL Dialogs provide powerful means of creating various dialogs and applications composed of, for example menus, buttons, lists, check boxes, graphs and tables. Visual SCIL dialogs are mainly used for various tools, control dialogs and so on.

Pictures are basically the classic way of illustrating the process known from SYS 500, but they are still supported for full backwards compatibility.

Visual SCIL Dialogs can be used together with both Process Displays and Pictures.

3.2.2 User Interface Design Methods

Pictures are designed and programmed in Picture Editor, see SYS600 Picture Editing. Process Displays are designed and programmed in Display Builder, see SYS600 Process Display Design. Visual SCIL Dialogs are designed and programmed in Dialog Editor and this process is described in this manual.

3.2.3 Using Process Displays and Dialogs

In this manual, the dialog concept is broader than usual. A dialog may be an operator dialog that informs the operator of something and requires a data entry or a click on a button. However, a dialog can also be a tool or a presentation containing a picture, an image, a list and so on. In some cases, the process displays and dialogs are equal alternatives when designing user-interface elements. In other cases, one of them is preferable. Generally, dialogs are suitable for:

- tools
- operator dialogs
- lists and textual reports
- tables

Process displays are more suitable for process illustrations that require dynamic graphical behaviour as well as zooming, panning and decluttering features.

3.3 Visual SCIL Design Method

3.3.1 Visual SCIL Objects

Everything contained in a dialog, for example, buttons, texts, menus, menu items, images, etc., are called dialog items. A text and a button are examples of dialog items, while a menu is a container group object as it generally contains menu items.

In SCIL, the dialogs and dialog items are handled as what is called Visual SCIL objects. Subsequently, the term Visual SCIL objects, or simply objects, will be used as a common name for dialogs and dialog items. In addition, it comprises images, which are not actually dialog items though they can be shown on dialog items.

3.3.2 Types of Visual SCIL Objects

There are approximately 40 types of Visual SCIL object types.

The Visual SCIL object types can be grouped into the following main types:

- Container group objects (dialogs and dialog items that can contain other dialog items).
    - Dialogs. Each entity in Figure 1 supplied with a title bar is a dialog (however, there can also be dialogs without title bars). There are two types of dialogs that look the same and have same functions but differ in the way attributes can be read from

other dialogs: main dialogs and dialogs. Also there are other types of dialogs, for example objects VS_FILE_CHOOSER and VS_NOTICE_DIALOG.
    - Containers. The containers are used for grouping dialog items. They can contain all kinds of dialog items.
    - Picture containers. A special type of containers, the picture container, can contain a picture.
    - Menus. The concept of menu as a container group object includes menu bars, menus and menu items.
    - Notebooks and notebook pages.
- Other dialog items: buttons, texts, labels, lists, etc.
- Images.

In SCIL and in the Dialog Editor the object types have names formed by adding a VS_ to the descriptive name of the type. The descriptive names will be used throughout this manual.

The Visual SCIL object types are detailed in SYS600 Visual SCIL Objects manual.

The Dialog Editor does not support the following Visual SCIL Objects:
VS_HEADER_ITEM
VS_NOTICE_DIALOG
VS_IMAGE_EDITOR
VS_FILE_CHOOSER
VS_PROGRAM_EDITOR
VS_INPUT_DIALOG
These Visual SCIL Objects must be inserted using a SCIL program.

3.3.3 Dialog Systems

A dialog system starts with a main dialog or a picture container. The main dialog and all dialogs opened from it, or from items included, belong to the same dialog system. Likewise, the picture container and all dialogs opened from the picture included in the container belong to the same dialog system.

Note that it is recommended to always use main dialogs when possible and form several dialog systems, because each dialog system has its own SCIL context i.e. its own variable space, object name space, etc.

3.3.4 Composing Procedure

The general procedure for composing a dialog system is:

- Designing dialogs and other objects in the Dialog Editor and storing them in a file. One dialog is the main dialog.
- Loading a main dialog from a main dialog or a picture (the main dialog is loaded automatically when a SYS600 monitor is started). A new main dialog starts a new dialog system.
- Loading other dialogs and objects from the main dialog.
  Figure 1 shows an example of a dialog system.

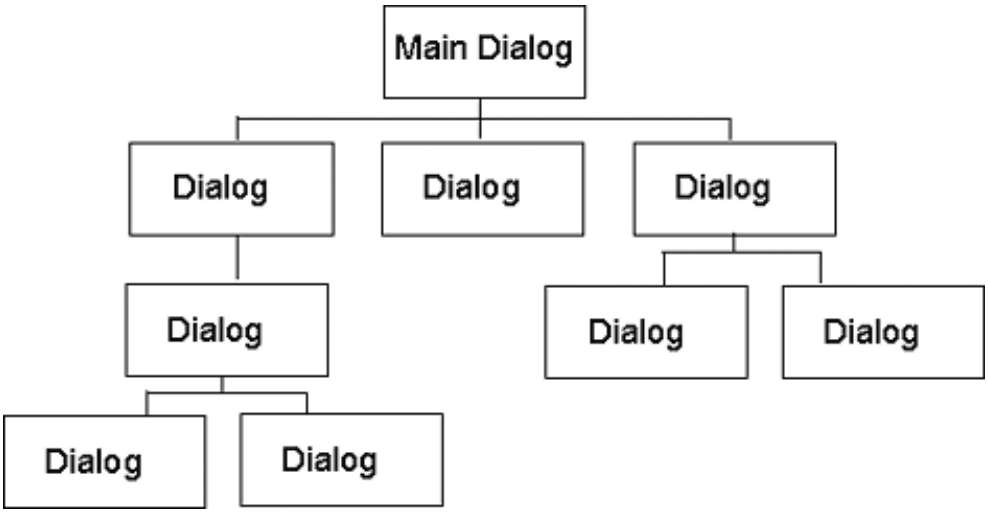


Figure 1: An example of a dialog system

3.3.5 Handling Visual SCIL Objects

The dialogs stored in files are loaded into a dialog system using the SCIL command .LOAD. A certain dialog can be loaded several times within the same or a different dialog system. This has to be done in such a way that an already created dialog has to be deleted before reloading it. A dialog is deleted from the dialog system with the .DELETE command.

Dialogs and dialog items can also be created with a SCIL command, the .CREATE command.

Each Visual SCIL object (dialog, dialog item, image) has an object name used for referencing the object in SCIL.

The features of the dialog and the dialog items are accessed as attributes and predefined methods. Examples of attributes are:

- Position and size
- Color
- Font
- Text contents

The attributes can be changed dynamically with the SCIL command .SET. Some features are affected by executing predefined methods.

3.3.6 Example

The dialog in the following example opens a new dialog if the operator clicks the **Yes** button. At the same time it deletes the present dialog. If the operator selects **No**, the dialog is deleted but no new dialog is opened. The method named NOTIFY is a user activated method activated by a click the object. The dialog is referred to as PARENT. ROOT refers to the main dialog or picture container of the dialog system.

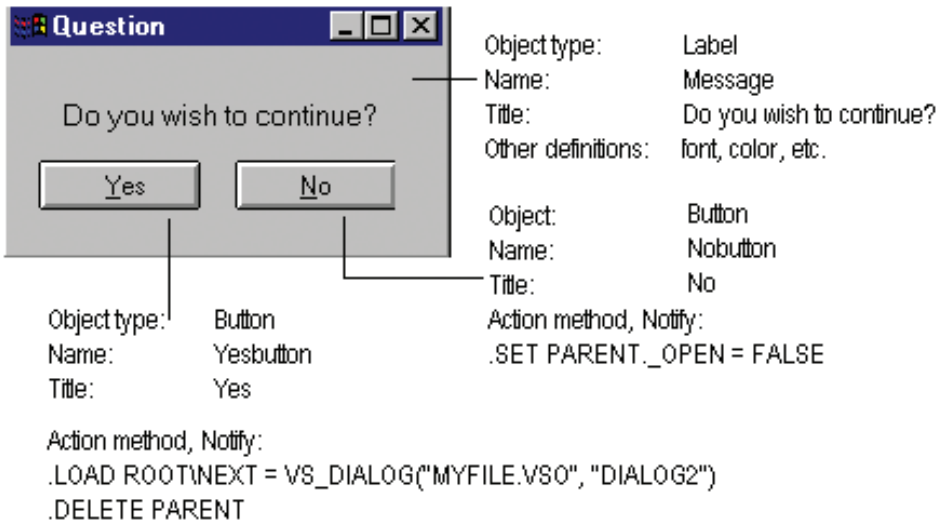


Figure 2: An example of a simple dialog and its methods

Section 4 Designing Dialog Systems

4.1 About this Section

This section describes the principles of how to design dialogs and other objects using the Dialog Editor. It also describes how to compose dialog systems by loading and creating dialogs and other objects. It contains the following sections:

Section 4.2	The principles for designing dialogs and other objects in the Dialog Editor. The rules for arranging dialog items within a dialog.
Section 4.3	Loading, creating and deleting dialogs and other Visual SCIL objects.
Section 4.4	The Visual SCIL object hierarchy within a dialog and within a dialog system. The initial main dialog.
Section 4.5	Coordinate system.

4.2 Creating Objects in Dialog Editor

4.2.1 General

By using the Dialog Editor the user can create main dialogs, dialogs, all types of dialog items and images and store them in files.

The Dialog Editor lets the user design dialogs, dialog items and images visually on screen using simple mouse operations. It also provides facilities for arranging dialog items neatly and connecting them to preserve their relative location and size when a dialog is resized.

4.2.2 Storing Objects

Normally, the user creates a dialog, adds contained dialog items and stores the dialog in a file, see Figure 3. However, all types of objects can be stored in files, from where they are loaded into the dialog system with SCIL, see Section 4.3. The objects stored by individual object definitions in a file can be loaded with SCIL. Objects contained within other objects cannot be loaded separately. One file can contain several object definitions.

In the Dialog Editor, each dialog item is defined by a name. This will be the Visual SCIL object name, which identifies the item within the dialog system. Object definitions stored in files are also given names. These names are used for identifying the items within the file.

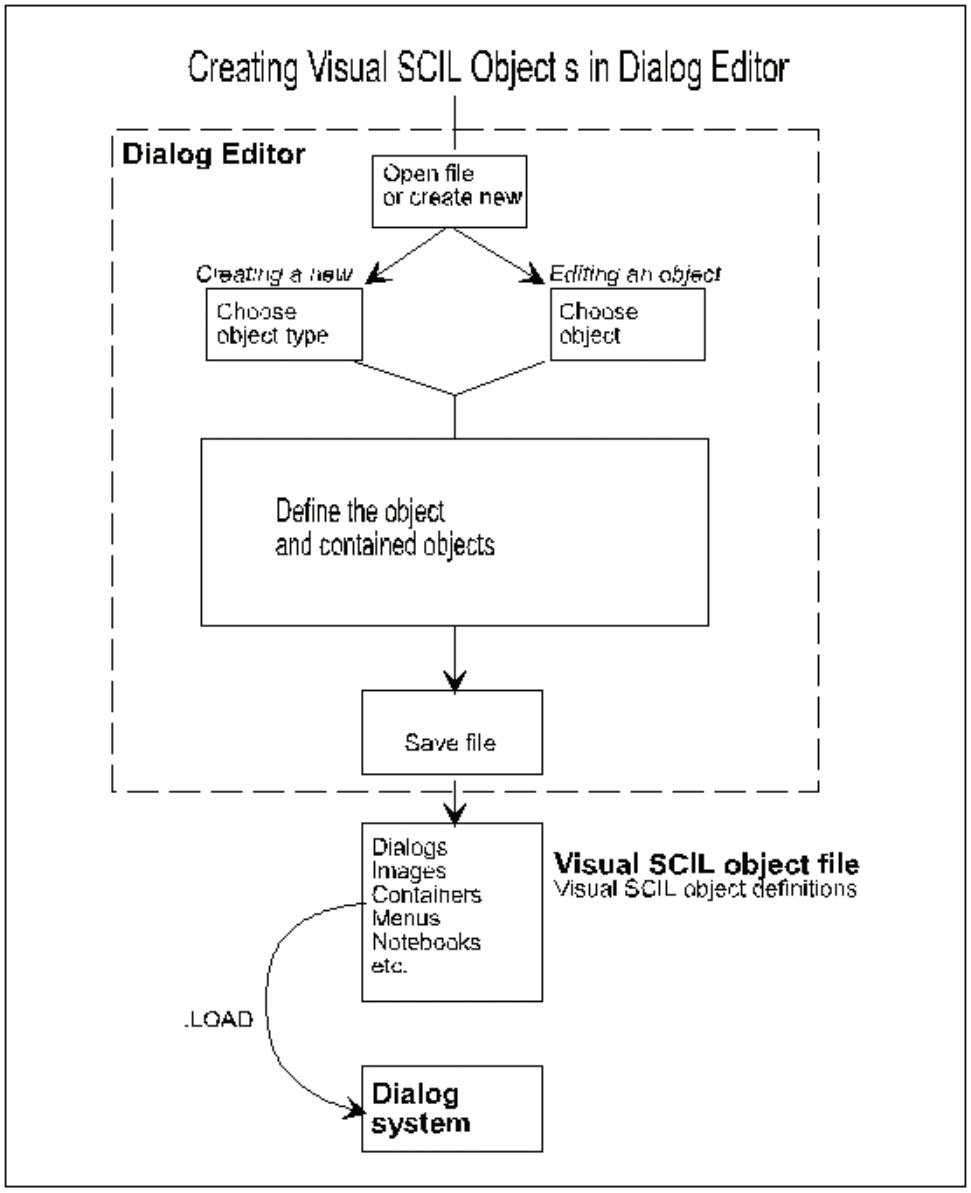


Figure 3: Creating objects in the Dialog Editor

The definition of Visual SCIL objects using the Dialog Editor is described in Section 6.

4.2.3 Designing Dialogs

A dialog may contain several dialog items, which in turn may contain dialog items. Figure 4 shows the rules for arranging dialog items within a dialog. Main dialogs, dialogs, containers and notebook pages can contain any type of dialog items. Menu bars can contain menus, which in turn can contain other menu items.

By setting some attributes of the Visual SCIL objects (see Section 5), the dialog items can be hidden and shown as well as disabled and enabled during operation.

Dialog items can also be loaded into a dialog and deleted from a dialog during operation using SCIL commands, see Section 4.3.

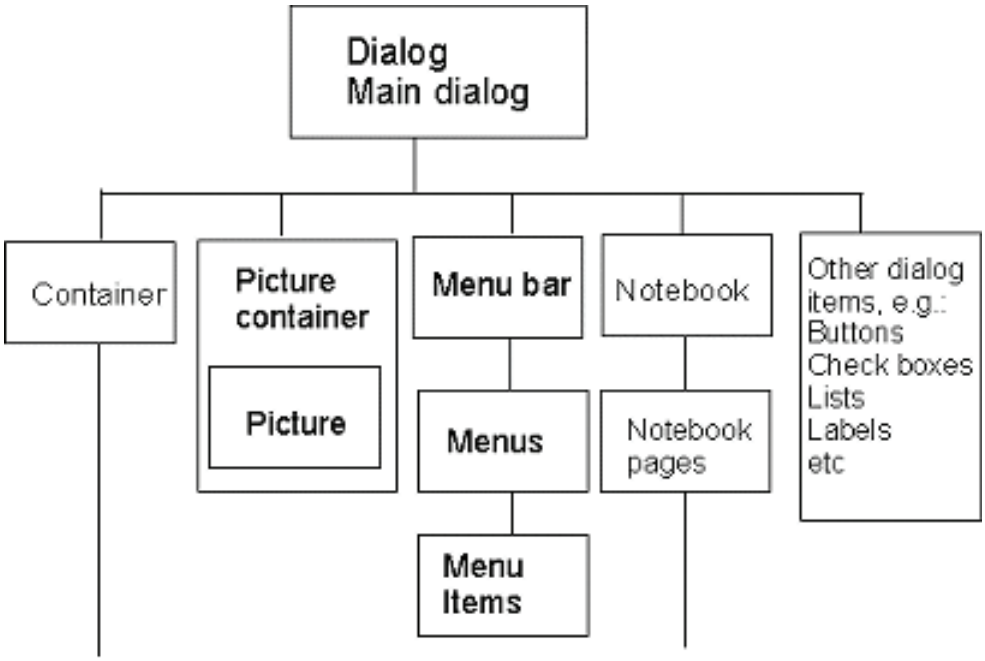


Figure 4: The possibilities to arrange dialog items within a dialog

4.3 Composing Dialog Systems

4.3.1 General

Loading and creating dialogs and other Visual SCIL objects with SCIL composes dialog systems.

A dialog system begins with a main dialog or a picture container. The main dialog and all dialogs opened from it, or from items included, belong to the same dialog system. Likewise, the picture container and all dialogs opened from the picture included in the container, belong to the same dialog system. The dialog system changes dynamically when new dialogs are loaded or dialogs are deleted. See Figure 5.

Usually, the type of a dialog is a main dialog. The difference between main dialogs and dialogs is that when a dialog is used, its attributes can be read from its parent dialog. The dialog and its parent are in the same SCIL context. If a main dialog is used, the variables cannot be used outside of it. Hence, no confusions between attributes of different dialogs occur. This is the reason why using main dialogs and several dialog systems is recommended always when possible.

The use of main dialogs is not recommended when the user wants to read the variables from other dialogs. An example of this is the Error Viewer and the Test Dialog opened from it. The main dialogs were not used while they were designed, because the user is shown information of SCIL errors that occur in other tools. The examination of variables in the tool where the error occurred is also supported in the Test Dialog.

All objects within the same dialog system have their variables in common. Visual SCIL objects and attributes are accessed by all objects within the same dialog system. Outside, only the objects in the interface between two dialog systems can access one another (see Section 5).

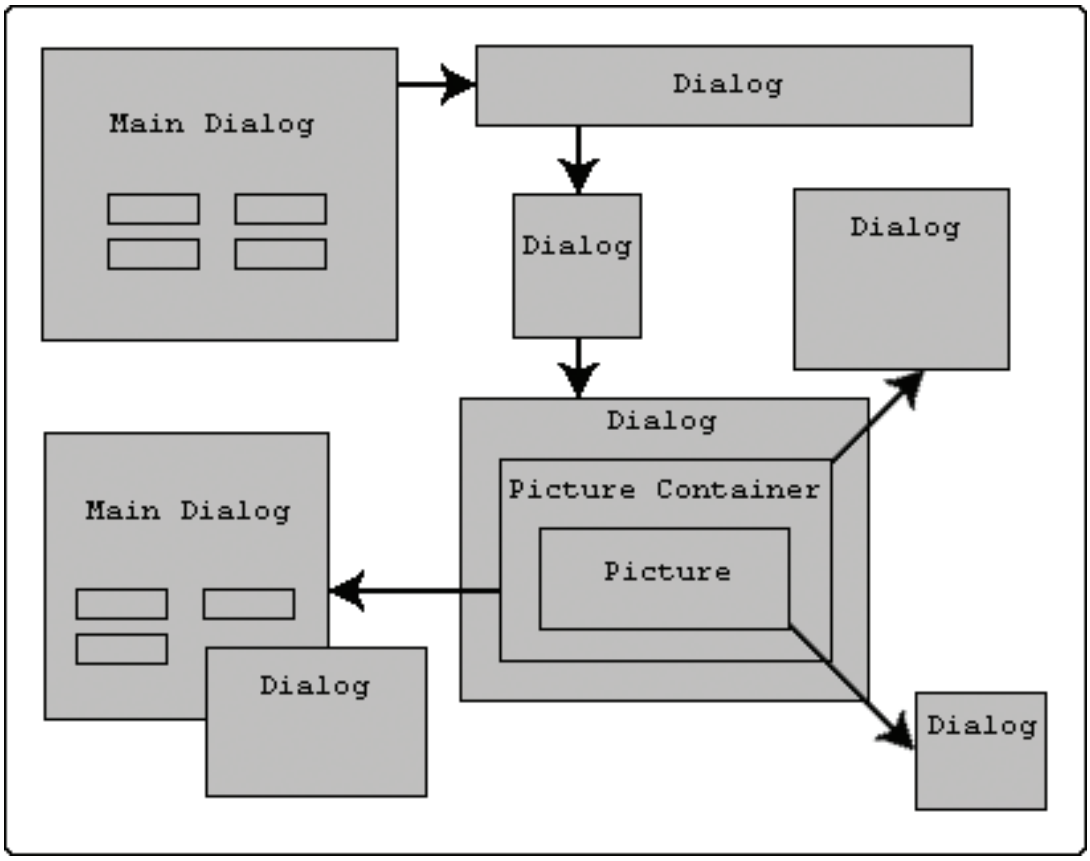


Figure 5: Three different dialog systems

4.3.2 Loading, Creating and Deleting Dialogs

To load a dialog, stored in a file, into the dialog system, use the SCIL command .LOAD. The command can be used in any method of any object in the dialog system (see Section 5). Dialogs can also be loaded from pictures.

When a dialog is loaded, all its dialog items are loaded as well. With the .LOAD command the loaded dialog is given a Visual SCIL object name, which is its identification in SCIL. The dialog items will be referenced with the names given in the Dialog Editor.

With the load command, the dialog may also be given attributes. For instance, the position of the dialog can be given by an attribute. This method allows the stored dialogs to be loaded in several situations and contexts without limitations. Hence, a dialog built in the Dialog Editor can appear simultaneously on several locations on screen, with same or different appearances (specified by attributes).

Dialogs can also be created with a SCIL command, the .CREATE command. The .CREATE command is used mainly in cases where some dynamic feature affects the composition of the object. It is also used for creating objects of some special ready made object types, which cannot be accessed from the Dialog Editor (see SYS600 Visual SCIL Objects).

To remove loaded and created dialogs from the dialog system, use the SCIL command .DELETE. The user can also hide and disable dialogs by using an attribute, see Section 5.

The .LOAD, .CREATE and .DELETE commands are detailed in SYS600 Programming Language SCIL.

4.3.3 Loading Dialog Items

In the same way as dialogs, objects of other types can be loaded, created and deleted. Dialog items can be loaded into dialogs, containers and notebook pages. Generally, however, this is not recommended as it might destroy the geometry management in the dialog.

4.3.4 Image Objects

As mentioned earlier, image objects cannot exist as dialog items. Images are always stored in files and loaded into the dialog system with .LOAD. Images are displayed on dialog items by assigning their image attribute the name of the image object or by giving the object names as argument of predefined methods. Some types of objects (boxes, buttons, containers) can also be given images in the Dialog Editor. These images do not have any names and they are not seen as objects.

Images can be drawn using the Image Editor of the Dialog Editor, which also allows importing images.

4.4 Object Hierarchy

4.4.1 Dialog Item Hierarchy within a Dialog

A dialog and the dialog items contained in it from a hierarchical object structure. An object contained in another object is the child of this object, which is called parent. The words ancestors and descendants can be used to describe indirect hierarchical relationships.

Figure 4, Figure 5 and Figure 6 illustrate a hierarchy. The dialog GROUP is the parent of the containers PUSH and GROUP. The container PUSH has three child objects: OK, CANCEL and HELP. The main dialog GROUP is an ancestor of HELP and HELP is a descendant of GROUP.

This parent child relationship also applies to dialogs and objects, which are loaded into the dialog system. See Figure 6.

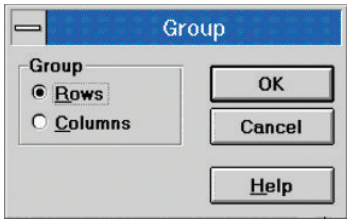


Figure 6: An example of a dialog object containing objects in two levels. The dialog contains three buttons and two option buttons. The option buttons are placed in a container with label Group.

4.4.2 Object Hierarchy within Dialog Systems

In a dialog system, the Visual SCIL objects form a hierarchical system in accordance with how they are loaded. On the top is always a main dialog or picture container.

The parent-child relationship is important for the use and display of dialogs and dialog items. The child objects depend on their parents, so that if a parent object is deleted, all the child objects are deleted as well. The hierarchy is also important when programming the dialogs, because the Visual SCIL object references depend on the hierarchy.

When a main dialog or picture container is loaded, it starts a new dialog system. Hence, the main dialogs and picture containers constitute the interface between different dialog systems. They can be referred to as the ROOT object (see Section 5).

Objects created with .CREATE obey the same hierarchy rules as objects loaded with .LOAD.

4.4.3 Initial Main Dialog

When a SYS600 monitor is started (a logical monitor of type VS is opened), the initial main dialog of the application is loaded. All other dialogs and dialog items are descendants of this object. Therefore, the initial dialog must be present the whole time the monitor is open, because closing it means that all child objects are erased from screen. The SYS600 monitor is open until this object is deleted.

If not otherwise defined in the text file Apl_Def.txt in the APL directory, the application uses the Tool Manager as initial main dialog. To use another dialog than the Tool Manager as initial main dialog, edit the apl_def.txt file as follows:

1. Remove the comment signs before the block starting with:
   Open **Start** dialog as first visible object.
2. Type the name of the file, where the object is stored and the name of the dialog.

4.5 Miscellaneous

4.5.1 Coordinate System

The coordinate system of the Visual SCIL objects (unlike the SCIL coordinate system, see SYS600 Programming Language SCIL), has its origin (0,0) at the bottom left corner of the screen or the object, and the range is -32768 to +32768. See Figure 7. When handling dialogs, the origin of the coordinate system lies at the bottom left corner of the screen. When handling dialog items, the origin is in the lower left corner of the parent object (dialog, container or notebook page). One unit in the coordinate system corresponds to one pixel.

The coordinates can be viewed in the Dialog Editor. Coordinates are also used, for example, when giving a position with the _GEOMETRY attribute and when reading a position with the _GET_POINTER_POS method. For more information about object position, see Section 7.

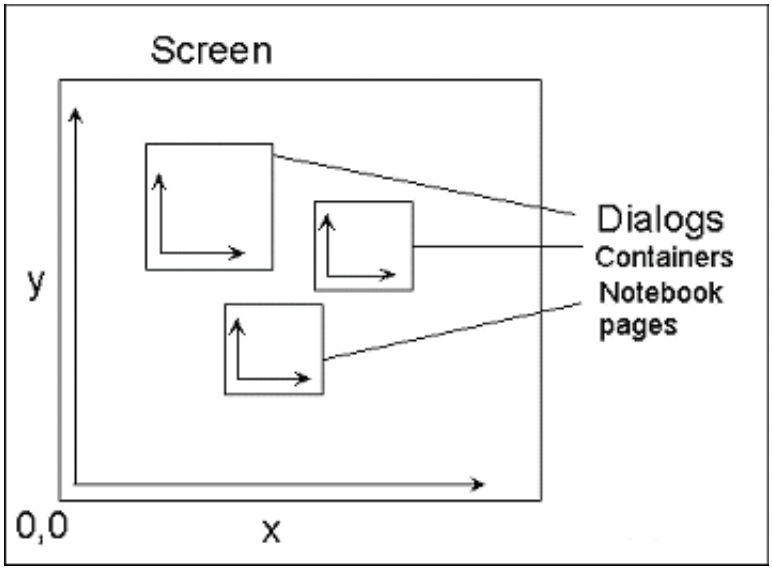


Figure 7: The coordinate system of the SYS600 dialogs

Section 5 Programming Dialog Systems

5.1 About this Section

This section describes how to program the dialogs and dialog items. It contains the following sections:

Section 5.2	This section provides an overview of the method types and their use.
Section 5.3	This section describes the attributes. Some important attributes, common to most objects, are discussed here.
Section 5.4	This section describes how to reference objects, how to reference and use attributes in SCIL, and how to use method calls in SCIL.
Section 5.5	Here you are given guidelines for programming dialog systems. Handling main dialogs, picture containers and language dependent texts are discussed.

5.2 Methods

5.2.1 Overview

The dialogs are programmed using a number of SCIL programs, which are called methods. The methods define the dynamic operation of the dialogs and dialog items. Each dialog and dialog item has its own set of methods. There are methods executed cyclically, methods executed on a certain user operation and methods executed on a certain event, etc. Some methods can be executed by a method call in SCIL.

Methods, except for some predefined and pre-programmed methods, are SCIL programs. Methods can contain all SCIL elements, except picture handling commands. Some predefined and preprogrammed methods are not editable. For more information regarding the rules for SCIL programs, see SYS600 Programming Language SCIL.

The methods are programmed in SCIL Program Editor, which is accessed from Dialog Editor. Generally, it is not possible to read or write the contents of the methods from SCIL. Only action methods can be read and written in SCIL via an attribute.

Each dialog and dialog item may have the following methods:

- A number of cyclic methods started repeatedly at a chosen time cycle.
- A number of event methods started automatically as a consequence of a chosen process event or SCIL event (event object).
- A number of action methods started at certain user operations, for example clicking a button.
- A Create method executed when the object is loaded, an Init method started after the create method and a delete method executed when the objects is deleted.
- An Error handling method started at a SCIL error in the object or its child objects.
- A Help method started when help is requested on the object.
- A number of predefined and preprogrammed methods which can be started with SCIL.
- A number of Arbitrary-user defined methods started with SCIL.

In the following descriptions, the methods are grouped according to the way they are activated.

5.2.2 Methods Activated Automatically

5.2.2.1 Cyclic Methods

Each dialog and dialog item may have a number of methods executed cyclically at a user specified interval (cycle time). The cycle time is defined in seconds. One object can have several cyclic methods with different cycle times.

The cyclic methods are executed the first time the object is loaded, after a possible Create method (see Section 5.2.4.1) and then cyclically until the object is deleted.

Do not use cyclic methods with short intervals, and do not start comprehensive programs cyclically.

5.2.2.2 Event Methods

Each object can also have an arbitrary number of event methods started at the activation of given Event Objects (see SYS600 Application Objects). The activating object is specified by entering the event object notation ('name':E'index') in the method definition.

5.2.3 Methods Activated by Operator

5.2.3.1 Action Methods

These methods are activated by certain user actions, such as clicking a button, dragging a scroll bar, entering data, etc. By programming the action methods in SCIL, the engineer defines how the objects will behave on user actions. Action methods have predefined names and activating operator actions, which are object type specific. Some object types have no action methods.

The names of action methods for the different object types and activating actions are listed and described for each object type in SYS600 Visual SCIL Objects.

The SCIL program of these types of methods can also be set and read from SCIL using a special attribute, see Section 5.3. For example, the object VS_BUTTON has the method NOTIFY, which is executed when the button is clicked. The object also has an attribute named _NOTIFY. The data type of this attribute is text vector and setting this attribute means that the NOTIFY method gets its contents from the SCIL program that was written to the attribute.

5.2.3.2 Help Method

The help method is executed when help is requested on the object containing the method. Help can be requested on a specific object in one of the following ways:

- By pressing the F1 key while the object has focus.
- From SCIL by calling the predefined method _GET_HELP of the object.

When help is requested, the first Help method that is found is executed. The Help method is first searched for in the object itself, that is, the object that has focus or the object from which the _GET_HELP method was called. If the method is not found, it is searched for in the parent

object. This search continues until the method is found or up to the ROOT object (main dialog or picture container).

The method has one input parameter that is the name of the object from which help is requested. The argument can be accessed using the SCIL function ARGUMENT. See SYS600 Programming Language SCIL.

Argument OBJECT_NAME has the name of the object on which help is requested.

If help was requested by pressing F1, the object name is the name of the object in focus. If help was requested by calling the _GET_HELP method, then the object name is the name of the object whose _GET_HELP method was called.

5.2.4 Methods Activated by Events in the Object

5.2.4.1 Create Method

The Create method is executed during the creation of an object after the .LOAD command has been issued. It is rarely needed, but can be used for initializing the attributes of the object.

When a dialog or dialog item is loaded, all its child objects are loaded as well. The create methods of the child objects are executed before the parent object. See the example in Figure 8. However, if there are several branches of dialogs and dialog items, the programmer cannot know which branch is executed first. In Figure 8, for example, the programmer cannot know what create method will be executed before: one of the CONTAINER and its child objects or one the of GROUP and its child objects. This should be regarded when programming the methods.

An object cannot be referenced until it has been loaded. The child objects are loaded before the parent object and the parent object can contain references to all its child objects. This means that the child objects can be referenced in the Create method of the parent objects. However, object references should generally be avoided in Create methods.

When the Create method of an object has been executed, the attribute assignments given in the .LOAD command are executed. Therefore, if the create method of the object contains attribute assignments, they may be overwritten by assignments in the .LOAD command.

If an error is encountered in a Create method, no object in the loaded structure is created.

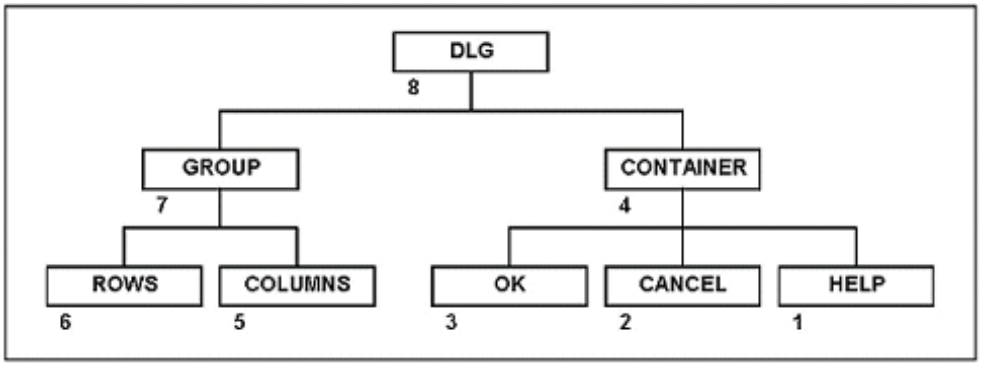


Figure 8: A possible execution order of the create methods (the numbered order) when the object DLG is loaded

5.2.4.2 Init Method

This method is started automatically after an object and all its parent objects have been created. The execution order of the Init methods of the object in the structure below could be as indicated in figure. That is, the Init method of the parent object is executed before the Init methods of the child objects.

Since all objects have been loaded before the Init methods are executed, all objects in the loaded structure can be accessed from the Init method.

The Create method of all loaded objects has already been executed when the Init method is executed.

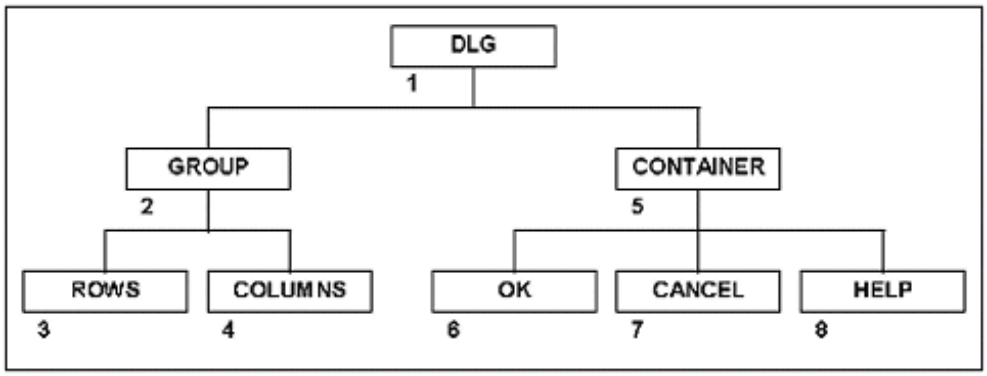


Figure 9: Possible execution order of the init method

5.2.4.3 Delete Method

This method is executed when the object is deleted. When several objects are deleted by deleting a common parent, the Delete methods are executed in the same order as for the Init method. Therefore, the delete method of the parent object (the object mentioned in the .DELETE command) is executed before the Delete methods of the child objects. An object cannot be referenced after its Delete method has been executed. Hence, the Delete method of the parent objects can reference child objects, but the Delete methods of the child objects cannot contain references to the parent object.

5.2.4.4 Error Handling Method

When a SCIL error occurs in an object, and the error handling policy is either CONTINUE or STOP, the error handling method is executed (see SYS600 Programming Language SCIL). If no such method is found in the object, it is searched for in the parent object. This search continues upward in the object hierarchy until an error handling method is found. The ROOT object is the last object that is searched. If no error handling method is found, a standard error dialog is shown.

When the Error Handling method is executed, it is automatically given the following six arguments, which can be used in the error handling program by the argument functions (see SYS600 Programming Language SCIL):

- STATUS SCIL status code (integer).
- METHOD The method where the error occurred.
- LINE SCIL line (text).
- POSITION Position on the line (integer).
- ERROR POLICY The error handling policy used when the error occurred, for example STOP or CONTINUE.
- LINE NUMBER The number of the erroneous line within METHOD.

The error handler method always has the error handling mode IGNORE even if the error state is explicitly set to something else.

5.2.5 Methods Activated by SCIL

5.2.5.1 Overview

Each dialog and dialog item may have an arbitrary number of user defined methods, which may be freely named and programmed. These methods are executed from SCIL by a method call. In addition, there may be a number of predefined methods. For more information on method calls, see Section 5.4.4.

5.2.5.2 Predefined Methods

Predefined methods cannot always be deleted or edited. The predefined methods, their names, purpose, activation, possible arguments and possible return values etc., are described in SYS600 Visual SCIL Objects.

The predefined methods have names starting with an underscore (_).

5.2.5.3 Arbitrary User Defined Methods

User defined methods may be private or public. Public methods are accessible by all objects within the dialog system, while private methods are accessible only within the same object and its child objects (its descendants).

In Dialog Editor, these methods can be found in the Public and Private pages of the object editors.

The user defined methods can be called with arguments. If arguments are given, they are accessed within the program using SCIL function ARGUMENT. They can also be programmed to return a value using the SCIL command #RETURN. See SYS600 Programming Language SCIL.

5.3 Attributes

5.3.1 Overview

Each Visual SCIL object can have a number of attributes, which specify the appearance and behavior of the object. The basic features given in Dialog Editor serve as default values when a dialog or dialog item is loaded. Most attributes are also accessible from SCIL and can be changed dynamically.

Each object has a number of predefined, object type specific attributes. In addition, an object may be given a number of user defined, object specific attributes.

5.3.2 Predefined and User Defined Attributes

5.3.2.1 Predefined Attributes

The predefined attributes have fixed names and meanings. The attributes are object type specific and cannot be deleted. When an object is created, its predefined attributes are given

default values, unless they are given other values in Dialog Editor or with the .CREATE or .LOAD command.

The predefined attributes of the different object types are detailed in SYS600 Visual SCIL Objects. It also describes the read/write access and the data type of the predefined attributes.

The predefined attributes have names beginning with an underscore (_).

5.3.2.2 User Defined Attributes

Every Visual SCIL object can be equipped with a number of user defined attributes for arbitrary purposes. These attributes are created by the SCIL commands .CREATE, .LOAD and .MODIFY. This is done by listing the attributes and their values in the argument list.

Examples:

```
.create CANCEL = VS_BUTTON(MY_ATTR=“abc”)
```

or

```
.modify CANCEL = LIST(MY_ATTR="abc")
```

or

```
.load CANCEL = VS_BUTTON(‘file_name’,‘obj_name’,MY_ATTR="abc")
```

Attributes cannot be created with the .SET command.

The data type of a user defined attribute is dynamic, hence the data type is determined by the value last assigned to the attribute. A new assignment may change the data type. The user defined attributes always have full read and write access.

5.3.3 Some Important Attributes

5.3.3.1 Visibility Attribute

The dialogs have an attribute that specifies whether it is shown or not. This attribute is named _OPEN and it can be either TRUE or FALSE.

Setting this attribute to TRUE means that the dialog with all of its child objects is opened on screen. Setting the attribute to FALSE means that the dialog with all of its contained objects is closed.

5.3.3.2 Enabled/Disabled

By setting the _ENABLED attribute to FALSE, the function of the object and all its child objects is disabled. This means that the action methods are disabled, but cyclic and event methods continue to be executed.

5.3.3.3 Position and Size

The position attribute, _GEOMETRY, specifies the position and size of the dialog or dialog item according to the coordinate system described in Section 4.

5.3.3.4 Attributes for Reading and Writing Action Methods

Objects with Action methods have special type of predefined attributes, which can be used to define the contents of the action methods from SCIL. The object type VS_BUTTON, for example, has the action method NOTIFY, which is executed when the button is clicked. It also has an attribute called _NOTIFY, which is a text vector. The content of the _NOTIFY attribute is the program of the NOTIFY method. By reading or writing to the _NOTIFY attribute, it is possible to read and write the NOTIFY program using SCIL.

All Action methods can be accessed with attributes.

Generally, this way of defining methods is used when the object is created from SCIL. In the Dialog Editor, the methods are written in the SCIL Program Editor. However, the _NOTIFY attribute can also be used when there is a need to change the method during operation.

Example:

```
.set OK._NOTIFY = vector(".delete DLG");writing the method
@program = OK._NOTIFY; reading the contents of the method
```

5.4 Handling Visual SCIL Objects

5.4.1 Overview

This section describes how to handle the Visual SCIL objects, their attributes and methods in SCIL. The Visual SCIL commands for Visual SCIL object handling are detailed in SYS600 Programming Language SCIL.

5.4.2 Loading, Creating and Deleting Objects

The following SCIL commands are used for loading, creating and deleting objects:

- .CREATE Creates an object.
- .LOAD Loads an object stored in a file. The name of the file and the name of the object within the file are given as arguments.
- .DELETE Deletes an object and its child objects.

The commands are detailed in SYS600 Programming Language SCIL.

5.4.3 Referencing Objects in SCIL

5.4.3.1 General

Object references are used when loading, creating and deleting objects, and when referencing attributes and methods. In addition, some attributes can be assigned object references as values.

The hierarchy of the objects within a dialog system was described in Section 4. The hierarchy is important when referencing objects.

5.4.3.2 Object Names

When a dialog or dialog item is loaded with the .LOAD command, it is assigned a Visual SCIL object name used to refer to it in SCIL. Child objects loaded within a parent have the Visual

SCIL object names given in the Dialog Editor. Likewise, visual SCIL objects created with the .CREATE commands are given an object name.

5.4.3.3 Object Paths

When referencing an object that is not directly one level below in the object hierarchy, a path is needed. An object path is the route to the referenced object seen from the own object's point of view. The objects passed on the route to the referenced object indicate the path. Each object name in the path is separated by a backslash (\) in dialogs and a stroke (/) or a backslash in pictures.

A path of object names can generally only be given downwards. To give a path to an object above the own object, the user must use one of the predefined path names ROOT or PARENT.

For example, the path from GROUP to the **OK** button in Figure 10 would be:

PUSH\OK

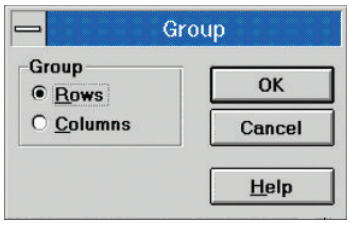


Figure 10: An example of a dialog object containing objects in two levels. The dialog contains three buttons and two option buttons. The option buttons are placed in a container with label Group.

If no path is given, the object is first searched for among the children of the present object. If there is no object with the given name, it is searched for among the children of the first child of the present object and then among the children of the second child.

For example, if the command

```
.set ROWS._TITLE = "Rows"
```

is encountered in the object GROUP in Figure 10, it entails a search for the ROWS object among the child objects SELECT and PUSH. As no objects are found there, the search continues on the next level where the object ROWS is found.

Wildcards can be used for any intermediate object level by using double separator signs.

Example:

```
.set DLG\\ROWS._TITLE = "Rows"
```

5.4.3.4 Predefined Object References

An object path can begin with one of the following three predefined object references:

- THIS The object in question, that is, the object containing the reference. Although this reference is generally not required, it is recommended because it gives a clearer program code and may help the programmer to avoid errors.
- ROOT The highest object in the hierarchy, that is, the nearest upper main dialog or picture container.
- PARENT The parent of the object where the reference occurs.

Example:

The **OK** button in Figure 10 could be referenced from SELECT by the following path:

PARENT\PUSH\OK

5.4.3.5 Remark to Predefined Object References

When a window (or a VS object) PARENT or THIS is referred to within a picture container context, the picture is first searched for a real window (or VS object) by name PARENT or THIS. It is used if found. If not found, the special meaning is applied (THIS refers to the current window, PARENT refers to the logical parent). This means that THIS and PARENT are not reserved words but can be used for names of VS objects although it violates the naming convention of VS objects.

5.4.3.6 Object Access

Using object paths, objects can be accessed from all other objects within the same dialog system. In addition, the main dialog objects and the picture container objects can be accessed from their parent objects, and the main dialogs and picture containers can access their parent objects. See Section 5.5.

5.4.4 Handling Attributes in SCIL

5.4.4.1 General

Using attribute references, the attribute values can be read and written from SCIL. Reading an attribute means that it is used in SCIL expressions. Writing an attribute means that it is assigned a value.

5.4.4.2 Attribute Names

The predefined attributes have predefined names, which are listed for each object type in SYS600 Visual SCIL Objects. The user defined attributes can be given freely chosen names of up to 63 alphanumeric characters.

5.4.4.3 Attribute Reference

Attributes are referenced with the following notation:

```
[object].attribute
```

where [object] is an object reference and attribute is the name of the attribute.

Attributes belonging to the object in question are referenced by attribute name only or by the object reference THIS.

Example:

```
THIS._TITLE
```

see the _TITLE attribute of the object in question.

5.4.4.4 Using Attributes in SCIL Expressions

The attribute reference can be used in SCIL expressions where the value of the attribute replaces the reference. The data type of the attribute determines how the attribute can be used in expressions.

5.4.4.5 Assigning Attributes Values

When an object is created, its predefined attributes are given default values which they have until changed. Attributes can be assigned values as follows:

- With the create command (.CREATE or .LOAD).
- With .MODIFY in create and init methods.
- With .SET in other methods.

The attribute values given in the create command overwrite values given in Dialog Editor or the default values. Values given in the Create and Init methods overwrite the values given in the create command.

Examples:

When setting attributes with the .CREATE, .LOAD and .MODIFY commands, the order of the attributes may be of importance as the following example shows:

```
.create DLG = VS_DIALOG(_GEOMETRY= LIST(X=100, Y=200, W=100, H=100),
_OPEN=TRUE)
.create DLG = VS_DIALOG(_OPEN = TRUE, _GEOMETRY= LIST(X=100, Y=200,
W=100, H=100))
```

The first line would result in a dialog appearing at X position 100 and Y position 200. The second line would result in a dialog first being displayed at default position (0,0), then jumping to the X position 100, and at last jumping to the Y position 200.

```
.set dlg._background_color = "red"
```

The statement defines that the background color is red.

```
.set PARENT\OK._TITLE = "Yes"
```

Changing the title of the OK object.

5.4.4.6 Attribute Access

The predefined attributes are read only, write only or read and write access allowed. They are accessible by all objects within the same dialog system. In addition, the attributes of main dialogs and picture containers are accessible by their parent objects. The user defined attributes have always both read and write access.

5.4.4.7 Data Types

The attributes can have any of the SCIL data types described in SYS600 Programming Language SCIL. The data types of the predefined attributes are given in the attribute lists in SYS600 Visual SCIL Objects.

The data types of the user defined attributes are not fixed but they get the data type of the value assigned to them. When first created, they get the data type of the value given in the create command. If the value is changed with .SET, the data type of the attribute may be changed as well. The data types of the predefined attributes cannot be changed.

The data types given in the attribute descriptions in SYS600 Visual SCIL Objects are the same as the SCIL data types, see SYS600 Programming Language SCIL. In addition, among the data types given in the lists, there are two data types that are not actually data types: Color and Font. Attributes with these "data types" are assigned values according to the color and font definition rules described in SYS600 Programming Language SCIL.

5.4.5 Handling Methods in SCIL

5.4.5.1 General

The arbitrary user defined methods and the predefined methods can be called from SCIL, which means that the methods are executed. Depending on whether the methods are public or private (see Section 5.2) they are accessible in the entire dialog system or only within the object.

A method call may be used as a SCIL statement or, provided that it returns a value, as an operand in SCIL expressions.

5.4.5.2 Method Calls

Method calls have the following format:

```
[object].method [(argument_list)]
```

where:

- [object] is an object reference. Not needed when referencing methods in the same object or one level below.
- method is the method name.
- (argument_list) are the arguments, which may be any SCIL expressions given as a list of SCIL expressions separated by comma and enclosed in parentheses. Up to 32 arguments may be given.

If the method returns a value, the method call can be used in expressions, and the returned value replaces the method call.

Examples:

LIST.UPDATE: This method takes no arguments and returns nothing. It just updates a list for example reading the list items from another location.

LIST.HIGHLIGHT(1): This method takes one argument that is the index of the item in the list. The LIST object then highlights this item and no data is returned.

A method is always executed in the context of the object in which it is defined

5.4.5.3 Using Method Calls in SCIL Expressions

A method (both predefined and user defined methods) that returns a value can be included as operands in SCIL expressions. The use of the method call depends on the data type of the returned data.

Examples:

```
@COLOR = BUTTON.PARENT_BG_COLOR
;The return value of the method is put in the variable COLOR.
```

```
@ITEMS = LIST.GET_ITEMS
;This method takes no arguments but it returns all items of the list to
the
;calling object as a vector. The vector is assigned to the variable ITEMS.
@ITEM = LIST.GET_ITEM(1)
;This method takes one argument, the index of the list item. The list
item ;found at this
    ;index is also returned to the caller.
.SET LABEL._TITLE = DATA_TYPE(BUTTON.PARENT_BG_COLOR)
;The data type of the background color definition of BUTTON's parent is
written ;to the
   ;title attribute of the object LABEL, for example to be displayed in
a ;dialog.
```

It is not possible to conclude from the syntax whether PARENT_BG_COLOR is an attribute or a method.

Implementation of the method PARENT_BG_COLOR:

```
#RETURN (PARENT._BACKGROUND_COLOR)
```

5.5 Programming Hints and Notes

5.5.1 General

This section provides hints and notes for the programming of dialog systems:

- Notes related to main dialogs and picture containers.
- Accessing objects outside the dialog system.
- Using language dependent texts.

5.5.2 Main Dialogs and Picture Containers

5.5.2.1 Initializing Main Dialogs

The main dialogs have some attributes by means of which the context of the object is initialized. The attributes are:

- COPY_LOGICALS: Copies temporary logical path's and representation libraries. Temporary logical path's and representation libraries are defined using the + sign (#path my_path +/sc/apl/test/my_pict). The default value for this attribute is FALSE.
- DEFAULT_PATH: Defines the path definition that is used when no logical path is defined in the main dialog context. This path is used when a file is referred to without path. The DEFAULT_PATH attribute must be given with the .LOAD command.

5.5.2.2 Handling Picture Containers

The picture containers have some attributes and methods for picture handling.

Attribute:

PICTURE_NAME Read only attribute that indicates the name of the picture currently shown in the picture container.

Methods:

NEW_PIC (name) Shows the picture 'name' in the picture container. The name can be given as a simple name, as path/name or as a full path name. The name is resolved in the caller's context.

LAST_PIC Works as !LAST_PIC.

INT_PIC Works like !INT_PIC.

SHOW_BACK (name) Shows the background of picture 'name' in the container.

The commands that change the picture (!NEW_PIC etc.) replace the contents of the picture container when executed within it.

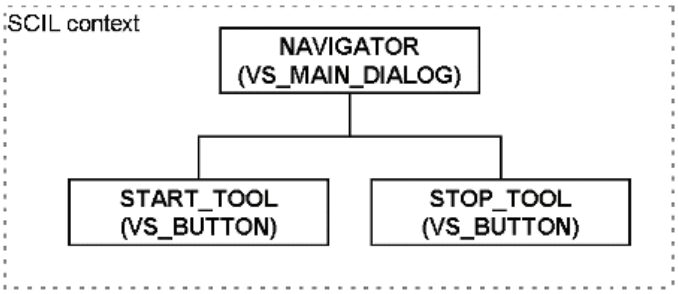
5.5.3 Accessing Objects Outside Dialog Systems

5.5.3.1 General

As a rule, the objects cannot be accessed outside the dialog system. However, the main dialogs and picture containers can access their parent objects, and thus can work as an interface against the neighboring dialog system. This is illustrated with two examples.

5.5.3.2 Main Dialog Example

Consider the following object structure:



Suppose that the START_TOOL button in figure above should load a TOOL object, which is a main dialog. The START_TOOL object contains the following NOTIFY method (action method started by a click of the button):

```
.load ROOT\TOOL = VS_MAIN_DIALOG('file','dialog')
```

In this case, the ROOT object seen from the START_TOOL object is NAVIGATOR. The TOOL object is hence created as a child of NAVIGATOR.

Setting the font of Individual tools after the .create or .load function may not load the tools with proper Font display, especially for bigger fonts. Font setting using _FONT is recommended during creation or load of the tool.

Example for Object Selector:

```
.load Application_Object_Selector =
vs_main_dialog("sys_tool/ObjNavig.vso","SUBMAIN",-
Operation_Mode = "SELECTION",-
Operation_Application = ROOT.APPL_NR,-
Operation_Object_Types = vector("T"),-
Operation_Filter = "",-
Selection_Policy = "ONE_ONLY_SELECTION",-
_title = translate("@l_Object_Selector"),-
_font = ._font,-
_icon = "ROOT\Product_Icon_M")
```

The figure below shows the resulting object hierarchy.

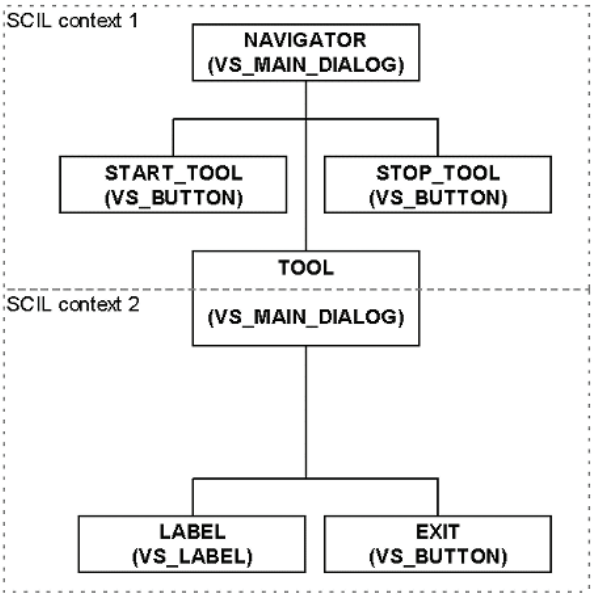


Figure 11: Object hierarchy

The objects in SCIL context 1 can now see the TOOL object, but not LABEL or EXIT. For instance, programs in context 1 can refer to attributes and methods in TOOL, but not in LABEL. The objects in context 2 cannot see any of the objects in context 1. The TOOL object can communicate with the NAVIGATOR only using the path names PARENT or ROOT. Consequently, the communication between the two contexts can only be accomplished indirectly using the PARENT and ROOT path names and the methods and attributes of TOOL.

Suppose that TOOL has a public method named EXIT_TOOL. The implementation of this could be:

```
;first exit functions like saving data etc.
.SAVE_DATA
.set ._OPEN = FALSE
.delete THIS
```

The button EXIT could have the following NOTIFY method implementation:

```
ROOT.EXIT_TOOL
```

The STOP_TOOL button could have the following NOTIFY method implementation:

```
ROOT\TOOL.EXIT_TOOL
```

5.5.3.3 Picture Container Example

Consider the following example:

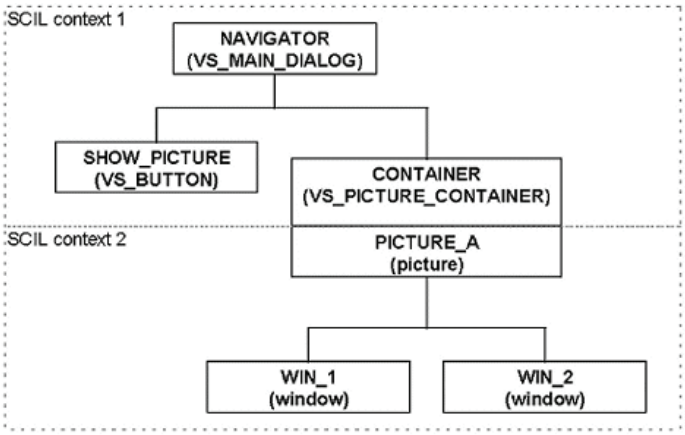


Figure 12: Picture container

The objects in SCIL context 1 can now see the CONTAINER object but not the objects inside it. The objects in context 2 cannot see any of the objects in context 1. The CONTAINER object can communicate with the NAVIGATOR only using the reference PARENT. Consequently, the communication between the two contexts can only be accomplished using the public methods and attributes of CONTAINER and NAVIGATOR. CONTAINER is the ROOT object in context 2. The windows and picture functions in the picture are child objects of CONTAINER.

5.5.4 Using Language Dependent Texts

5.5.4.1 General

The Dialog Editor provides tools for entering text identifications and translations to one or more languages. Text identifiers can be used in the Dialog Editor (for the Title attribute) and in SCIL. During operation, the text references are substituted with the corresponding texts of the selected language, see Figure 13.

The text references and the corresponding texts for different languages are specified in object specific text databases. The text reference is a SCIL identifier of maximum 63 characters. The language text databases are stored together with the dialog and dialog items. The text references can be typed using Dialog Editor or Language Editor.

The used language is defined by base system attributes and can be changed by a SCIL command and by a SCIL function.

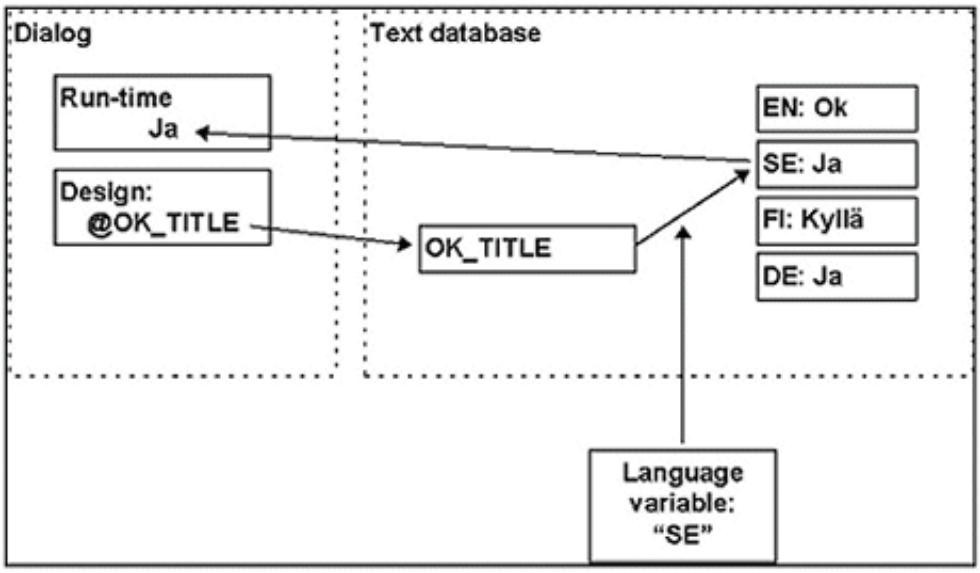


Figure 13: Language dependent text

5.5.4.2 Language Dependent Texts

Each Visual SCIL object can have its own set of language dependent texts. Due to the search mechanisms, it is recommended to store the language dependent texts in the object that will be loaded with SCIL.

When a text identifier is encountered, the reference is translated to the used language according to translations given in the text database of the object. If either the text reference or the language is not found in the object in question, it is searched in its parent object. If not found there, the search continues upward in the hierarchy towards the object that was loaded. The search continues until the text is found or the object that was loaded is searched through.

Example: If the structure below is loaded, a text reference encountered in the OK object is searched from OK, CONTAINER and DLG, assuming that it is not found in OK or CONTAINER.

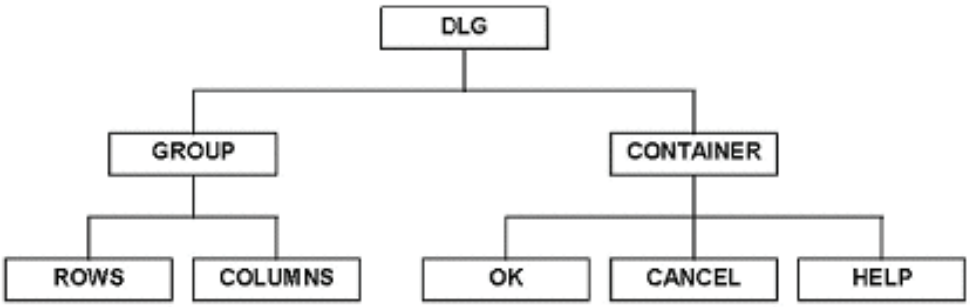


Figure 14: Text reference search

5.5.4.3 Text Identifiers

The text identifier is given as:

```
@name
```

where name is the name of the text reference.

In the Dialog Editor, the Title attribute can be given by a text identifier.

Text identifiers can also be used in SCIL using the TRANSLATE function, see SYS600 Programming Language SCIL. It is also possible to select another language than the one specified in the base system configuration.

If the user has regarded language versions when designing the dialog system, the TRANSLATE function has to be used to set texts dynamically.

Example:

```
.SET THIS._TITLE = TRANSLATE("@TITLE")
```

The statement sets the title of the object to the text with the text reference @TITLE and the chosen language.

Section 6 Using Dialog Editor

6.1 About this Section

This section describes the principles on how to use the Dialog Editor.

Section 6.2	The first section of this section describes the general principles of using the Dialog Editor. Opening and closing the editor is described along with the menu structure and an overview of the working procedure in Dialog Editor.
Section 6.3	The second section describes file handling procedures. An explanation of how to add, open, save and close files and of how to add and use templates is given.
Section 6.4	The third section describes object-handling procedures, for example, adding, selecting and renaming objects. Cut, copy, paste, duplicate, resize, align and delete functions are also described.

6.2 General

Dialog Editor is used to define VS (Visual SCIL) objects, such as dialogs, dialog items and images. In Dialog Editor the user can either add a new dialog or edit one that already exists. The working procedure, by which the user should add a new dialog and the Visual SCIL objects related to it, is described in this section. However, objects can be edited in any order.

In Dialog Editor the user can create VS object definitions, which are stored in files. The object definitions are loaded from the files into dialog systems. This is done using SCIL. Only one file at a time can be open in the editor. The user can either create a new VSO (Visual SCIL Objects) file or edit an existing one. There can be several dialogs and other VS objects in a file.

When a new dialog is defined, the process usually begins by adding the dialog itself, the VS_DIALOG or VS_MAIN_DIALOG object. Then the other objects that are part of the dialog, the dialog items, are added. In this manual, the VS object types are without the VS_ extension at the beginning of the name. For example, VS_MAIN_DIALOG is called main dialog.

6.2.1 Entering and Closing Dialog Editor

To open the Dialog Editor, double-click the Dialog Editor icon in the **User Interface** Page in the Tool Manager. The Dialog Editor Main Dialog appears. Continue by opening an existing file or creating a new file.

To close the Dialog Editor, choose **Exit** from the File menu of the Dialog Editor Main Dialog, or click the **Close** button in the upper right corner. To activate the Main Dialog, click the window titled Visual SCIL Dialog Editor or close all other windows. All editors of the Dialog Editor are closed.

6.2.2 Menu bar and Toolbar

The Dialog Editor is used with the help of menus like a Windows program. The menus are different in different editors, and even within the same editor when different pages are activated. The options that cannot be chosen at the moment are dimmed.

The menu bar and toolbar of the Dialog Editor Main Dialog are shown in Figure 15.

The menus in the menu bar are:

- The **File** menu, where the user can choose to open or close a file, add a new one or exit the Dialog Editor.
- The **Template** menu, where the user can choose to create a new template or use one that already exists.
- The **Tools** menu, where the user can choose functions related to testing, viewing the object tree or compressing a file.
- The **Options** menu, where the File History Length function gives the user a possibility to adjust the File menu history length between 0 and 20. Selecting 0 means that the file history is disabled.
- The **Help** menu, where the user can choose to read the About Dialog Editor information of the product.

The first button in the tool bar is the New button, works as **New File**/**Empty File** in the File menu. The second button, the **Open** button, works as **Open** in the File menu. The third button, the **Close** button, works as **Close** in the File menu. The fourth button, the **Test** button, works as **Test Selected Object** in the Tools menu. The fifth button, the **Stop Test** button, works as **Stop Test** in the Tools menu.

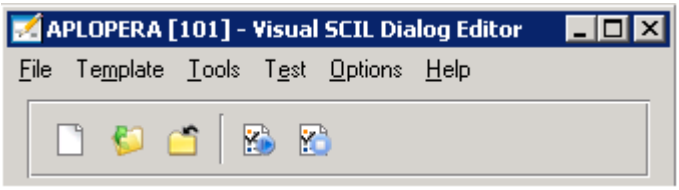


Figure 15: The menu and toolbars of the Dialog Editor Main Dialog

The Object List and object editors also contain menus. The menu bar of the Object List is shown in Figure 16.

The menus are:

- The **File** menu, where the user can choose to save a file or exit the Object List.
- The **Edit** menu, where the user can choose functions relating to moving, copying or deleting.
- The **Object** menu, where the user can choose to edit, add, rename or change the object type.

Figure 16: The menu bar of the Object List

The menu bar from the Dialog Object Editor is shown in Figure 17 as an example of object editor menu bar.

The menus in the Dialog Object Editor are:

- From the **File** menu the user can choose to close the dialog.
- From the **Edit** menu the user can choose functions relating to moving, copying, deleting, selecting and defining colors or fonts.
- From the **Dialog** menu the user can choose functions relating to adding and editing dialog items.

- From the menu the user can choose functions relating to graphic layers, aligning functions and distributing.
- From the **Connections** menu the user can choose how an object is connected to other objects. These functions help to change the size and shape of objects.
- From the **Windows** menu the user can choose to open a Connections Editor.

The menu bar differs according to the object being edited.

Figure 17: The menu bar of the Dialog Object Editor

6.2.3 The Working Procedure

The process of adding and defining an object is different depending on the object type and where the object should be created. An example of the workflow of adding and defining an object is shown in Figure 18.

Defining a dialog contains the following steps:

1. Open the Dialog Editor. The Dialog Editor Main Dialog is the first dialog box opened.
2. Open a file. Either a new file can be created or one that already exists can be created. The user can also create a template or use a template to create a file that is similar to previously created ones. The Object List appears.
3. Add a dialog. Choose either VS_DIALOG or VS_MAIN_DIALOG.
4. Add dialog items. The dialog items can be added in the Object List or in the Dialog Object Editor.
5. Define the objects. This is done in the object editors. All object editors have several notebook pages. To see each page, click its tab.
6. Save the file.

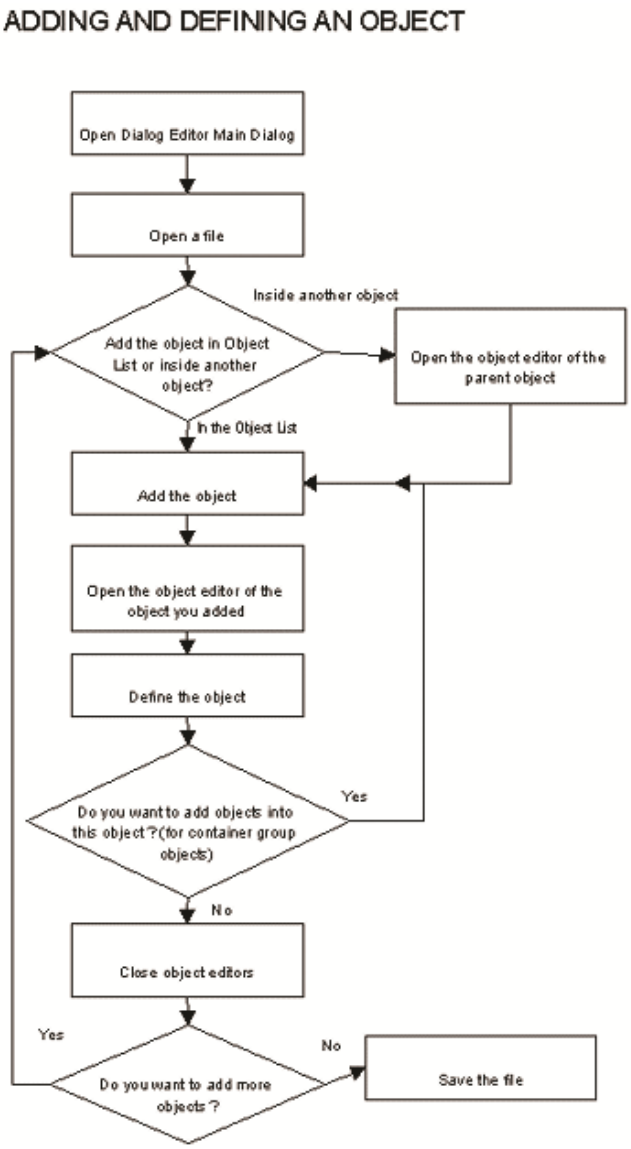


Figure 18: An example workflow of adding and defining an object. For information when to add the object in the Object List or in the object editor, see "Adding Objects" in Section 6.4.6. If you want to add more objects inside the same object, it is not necessary to close the object editor of that object.

6.2.4 Dialogs and Dialog Items

The Dialog Editor is mainly used for creating dialogs that contain dialog items. The other container group objects can be added into a dialog. They can contain dialog items. The Dialog Items are presented in Figure 19. The container group items in the picture are container, menu bar, notebook and picture container. Also, dialog belongs to the container group.

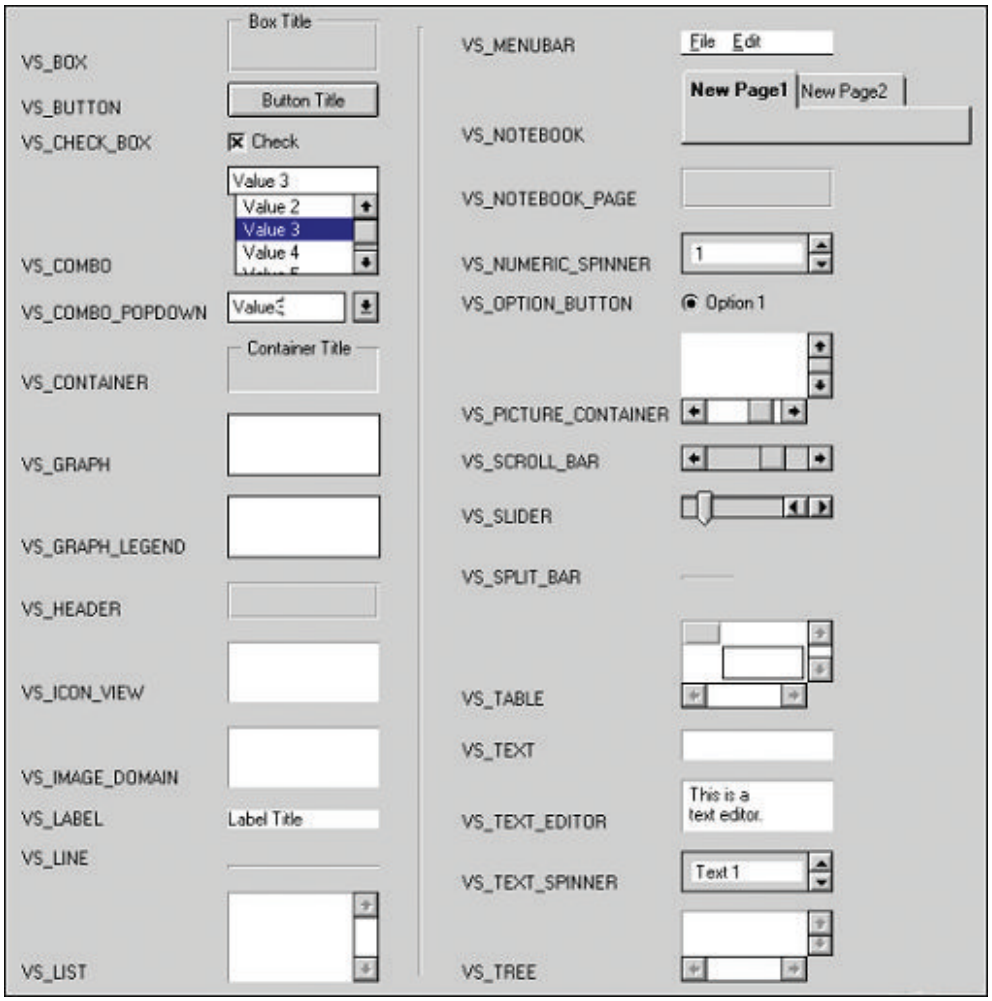


Figure 19: The Dialog Items

6.3 File Handling

6.3.1 Creating a New File

When entering the Dialog Editor, there are no files open. The user can either create a new one or edit one that already exists. Note that only one file can be edited at a time. If a file is already open, it is closed before the new file is opened. To create a new file:

1. Choose **New** from the **File** menu of the Main Dialog.
2. Choose to open an empty file or a main dialog template file. The main dialog template file contains a standard main dialog.

6.3.2 Main Dialog Template

A template for a main dialog is provided in the dialog editor which can be opened from **File/New From Template** menu item, this template is developed as part of the TPL_MAIN.VSO. The template contains a menu bar, a toolbar and some sample code, and it can be used as a base when a new tool is created. Runtime behavior for main dialog template will be as shown in Figure 20.

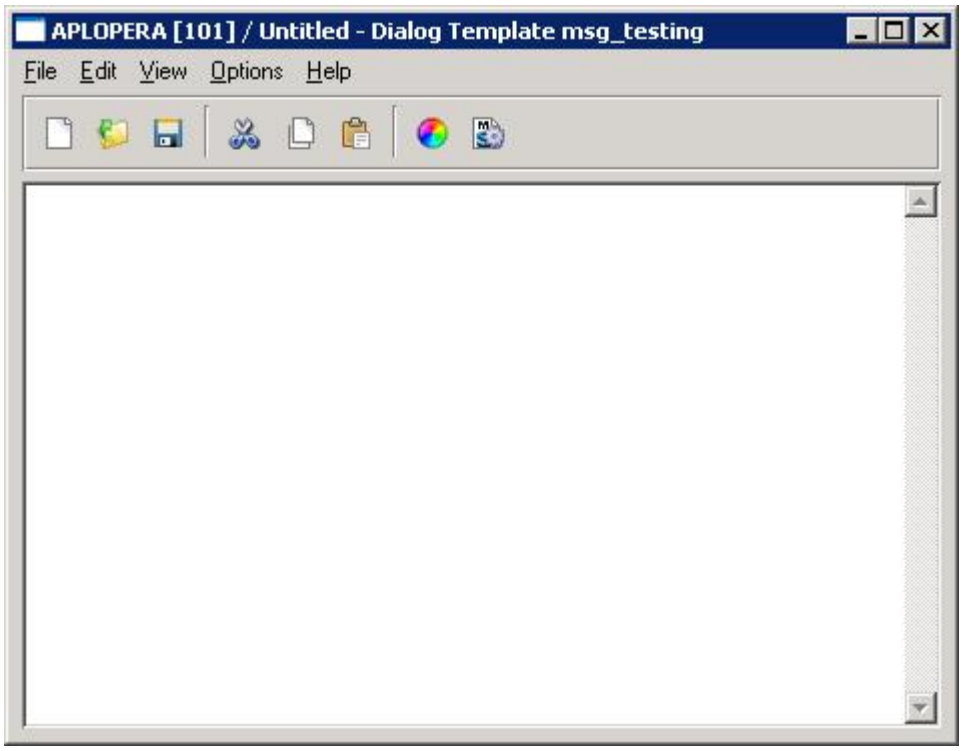


Figure 20: The main dialog template

The template contains sample code e.g. for the following functions:

- File operations
  1. Create a new file
  2. Open the existing file
  3. Updating/Saving of the existing file
- Text related operations, such as cut, copy and paste
  Example for Copy of text:

```
#if length(txt_File_Contents._get_selected_text) > 0 #then-
       .set ._clipboard_text = txt_File_Contents._get_selected_text
```

  Example for Cut of text:

```
txt_File_Contents._set_selected_text(vector())
    .set ._focus_item = "txt_File_Contents"
    txt_File_Contents._scroll_selection_into_view
```

  Example for paste of text:

```
#if ._clipboard_has_contents #then #block

       #if not txt_File_Contents._set_selected_text(._clipboard_text)
#then -
          .Show_Message(translate("@MSG_PASTE"),
translate("@MSG_PASTE_ERROR"), "CRITICAL")

    #block_end
```

- File menu with file history list
  This feature is mainly useful in maintaining the history of files opened in previous sessions and can be achieved using the following code.
  Example for Setting History:

```
@v_History = .v_File_History
```

```
    #error ignore
       .delete mnb_Menubar\mnu_File\mnr_File_History
       .delete mnb_Menubar\mnu_File\mni_File_Exit
    #error stop

    #if length(%v_History) > 0 #then #block
       #loop_with i = 1 .. length(%v_History)
          #error ignore
             @i_Status = status
             .set mnb_Menubar\mnu_File\mni_History'i'._title = "'i' "
+ %v_History('i')
          #error stop
          #if status <> 0 #then #block
             .create mnb_Menubar\mnu_File\mni_History'i' = vs_menu_item
             .set mnb_Menubar\mnu_File\mni_History'i'._title = "'i' "
+ %v_History('i')
             .set mnb_Menubar\mnu_File\mni_History'i'._mnemonic = "'i'"
             .set mnb_Menubar\mnu_File\mni_History'i'._notify =
vector(-
                "ROOT.on_File_Open(substr(._title, 3, 0))", "")
          #block_end
       #loop_end
       .create mnb_Menubar\mnu_File\mnr_File_History =
vs_separator_menu_item
    #block_end

    .load mnb_Menubar\mnu_File\mni_File_Exit =
vs_menu_item(._source_file_name, "mni_File_Exit")
```

- Storing settings between tool sessions
  This feature is useful in storing the setting between tool sessions, especially to store color/font settings and toolbar visibility. The following piece of code can be used as reference to perform this action.
  Example for Storing Settings:

```
.load mdl_Font_Chooser = vs_main_dialog(-
          .t_Font_Chooser_File, "MAIN",-
          Notify_Method = "Notify_Font_Chooser",-
          _leader_dialog = "ROOT",-
          _modal = ROOT._modal,-
          _icon = "ROOT\PRODUCT_ICON_M")

       mdl_Font_Chooser.Set_Selection("MIX")
       mdl_Font_Chooser.Set_Font("MIX", .l_Font)

       mdl_Font_Chooser._place("ROOT", "PLACE_CENTER",
"PLACE_CENTER")
       .set mdl_Font_Chooser._open = TRUE
       .set .b_Font_Chooser_Created = TRUE
```

- Showing messages and questions.
  This feature is useful in situations where a message is to be shown to users, or when a question is to be asked. The following piece of code can be used as reference to perform these actions.
  Example for showing message:

```
.create 'tObject' = vs_notice_dialog(-
          _title             = argument(1) + " - "
+ .product_description,-
          _icon              = "Product_Icon_M",-
          _notice_icon       = "ICON_'tMessage_Type'",-
          _text              = argument(2),-
          _modal             = true,-
          _ok                = (".set ._Open=false",".delete this"),-
          _cancel            = (".set ._Open=false",".delete this"),-
          _close_notify      = (".set ._Open=false",".delete this"))
```

```
    'tObject'._place("ROOT","PLACE_CENTER","PLACE_CENTER")
    .set 'tObject'._Open = true
```

Example for Showing Question message:

```
.create dlgQuestion = vs_notice_dialog(-
          _is_caution_dialog = true,-
          _icon              = "Product_Icon_M",-
          _title             = argument(1) + " - "
+ .product_description,-
          _text              = argument(2),-
          _has_cancel        = true,-
          _modal             = true,-
          _yes               =
("parent._queue_for_execution(%xYesCallback)",".set ._Open=false",".del
ete this"),-
          _no                =
("parent._queue_for_execution(%xNoCallback)",".set ._Open=false",".dele
te this"),-
          _close_notify      =
("parent._queue_for_execution(%xCancelCallback)",".set ._Open=false",".
delete this"))

    dlgQuestion._place("ROOT","PLACE_CENTER","PLACE_CENTER")
    .set dlgQuestion._Open = true
```

- Toolbar with standard icons, which can be visible or hidden.
  In view menu, ticking the Toolbar option will enable the toolbar, and not ticking the same will disable the toolbar.
  Example for Enabling/Disabling of Toolbar:

```
#if not .b_Toolbar_Visible #then #block
       .load nbk_Toolbar\nbp_Toolbar =
vs_notebook_page(._source_file_name, "NBP_TOOLBAR")
       .Set nbp_Toolbar._Enabled = true
       .Set nbp_Toolbar._Visible = true
       nbp_Toolbar._Select
       .Set .b_Toolbar_Visible = TRUE
       .Set mne_View_Toolbar._On = TRUE
       .Set_Command_Sensitivity
    #block_end
    #else #block
       #Error ignore
       .delete nbp_Toolbar
       #Error stop
       .set ROOT.b_Toolbar_Visible = FALSE
       .Set mne_View_Toolbar._On = FALSE
```

- Use of the standard color/font chooser.
  Example for Color chooser:

```
.load mdl_Color_Chooser = vs_main_dialog(-
          .t_Color_Chooser_File, "MAIN",-
          Notify_Method = "Notify_Color_Chooser",-
          _leader_dialog = "ROOT",-
          _modal = ROOT._modal,-
          _icon = "ROOT\PRODUCT_ICON_M")
```

  Example for Font chooser:

```
.load mdl_Font_Chooser = vs_main_dialog(-
          .t_Font_Chooser_File, "MAIN",-
          Notify_Method = "Notify_Font_Chooser",-
          _leader_dialog = "ROOT",-
          _modal = ROOT._modal,-
          _icon = "ROOT\PRODUCT_ICON_M")
```

- Toolbar with different size of icons.
  This feature is mainly used to set icon size the same as with monitor pro icon size. These icons will be available in three sizes: small, medium and large.

The locations of these icons are as below:

1. Small = sc\Stool\Misc\Images16.vso
2. Medium = sc\Stool\Misc\Images24.vso
3. Large = sc\Stool\Misc\Images32.vso

6.3.3 Opening an Existing File

There are two ways to open a file for editing. Firstly, it can be opened from the list of recently used files in the **File** menu. The length of the list can be changed in the **File History** dialog, which is opened from the **Options** menu. If the file name is not on the list, use the Open dialog box to open the file in the following way:

1. Choose **Open** from the File menu of the Main Dialog. In the File Chooser dialog box, files are chosen based on their name and location in the filesystem. Paths can be selected in four different modes that are listed below.
2. Find and click the name of the file to be opened and click **Open**.

If a read only VSO file is opened, the warning dialog box in Figure 21 is shown.

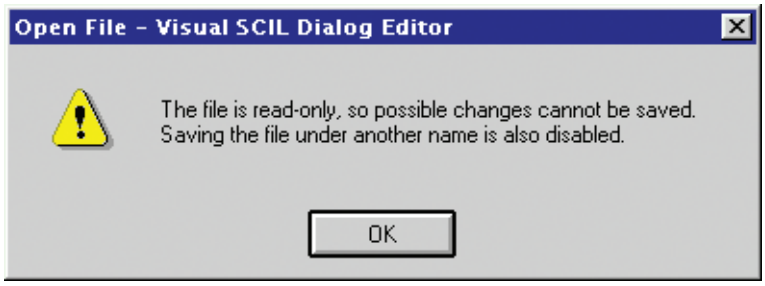


Figure 21: The dialog box warning about opening a read-only VSO file

In the File Chooser, paths can be selected in four different modes:

- Application Relative Paths
  Path representation in SYS600 path format relative to the current SYS600 application home directory. The application home directory itself cannot be referenced.
- SYS600 Relative Paths
  Path representation in SYS600 path format relative to the SYS600 root directory. The SYS600 root directory itself cannot be referenced.
- Logical Paths
  Path representation in SYS600 logical path format.
- Operating System Paths
  Path representation in the format used by the operating system.

The size of a file and the time of the last save operation are given in the File Chooser.

6.3.4 Breaking File Lock

The file the user wants to open might be locked. The reason for this might be that another person is editing the file or that the Dialog Editor has been exited abnormally. The dialog box shown in Figure 22 informs about the file lock and gives the options **Yes** or **No** for breaking the lock. The reason for file lock should be checked before breaking it.

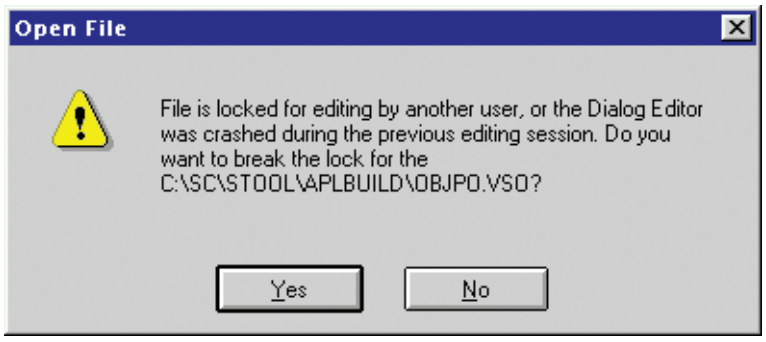


Figure 22: The dialog box that informs about a file been locked

If **Yes** is clicked, the file lock is broken, and the requested VSO file is opened. If **No** is clicked, the Open function is cancelled.

The file lock should not be broken if it is not absolutely necessary.

6.3.5 Saving a File

To save a file that has not been saved before:

1. Choose **Save As** from the File menu of the Object List. The Save dialog box appears.
2. Enter the filename for the file by clicking the text box and then type the name there. It is recommended to use maximum 10 characters in the name. The extension is added to the name automatically. Letters A-Z, numbers 0-9 and the underscore (_) can be used in filenames. Begin the name with a letter. The file is saved in the \pict directory, unless the location is changed by the user.
3. Click **Save**.

To save a file that has already been saved, choose **Save** from the File menu of the Object List or press CTRL+S. To save a file with another name, use the **Save As** function.

If a VSO file is a read only file, saving it is not possible, and the **Save** and **Save As...** commands in the File menu of the Visual SCIL Object List are disabled.

6.3.6 Compressing a File

When objects are added and edited in a file, the file size is increased, but if objects are deleted, the file size is not automatically decreased. Hence, the size of a VSO file can be gratuitously large unless space is released.

Compressing File releases space and reduces the size of the VSO file. To compress file, choose **Compress File** from the Tools menu in the Main Dialog.

6.3.7 Closing a File

To close a file, choose **Close** from the File menu of the Object List. If the changes that have been made to the file have not been saved, the user is prompted to do it now. If the file is a read only file and changes have been made, the Object List closes without giving the possibility to save the changes.

6.3.8 Using a Template

If similar objects are used in several files, it might be useful to create a template and use it when creating new files. A template is a file saved as a template. A template file cannot be edited, but objects can be copied in it and they can be inserted into another file. This way, the same definitions do not need to be made in all new files. To open a user defined template:

1. Choose User from the **Template** menu. Unless there are previously saved templates in the system, the User option is grayed out. Therefore, a template has to be created first.
2. On the appearing submenu, choose the template to be used. If the desired template is not on the menu, it has to be created first.
3. Open another file.
4. Select a dialog, main dialog, container or notebook and open its editor.
5. Select the dialog item in the picture of its parent object.
6. Copy objects from the template to the file.

6.3.8.1 Adding a Template

To add a user template:

1. Choose **Define User Templates** from the Template menu. The dialog box shown in Figure 23 appears.
2. Click **Add** and the name of a new template appears in the list.

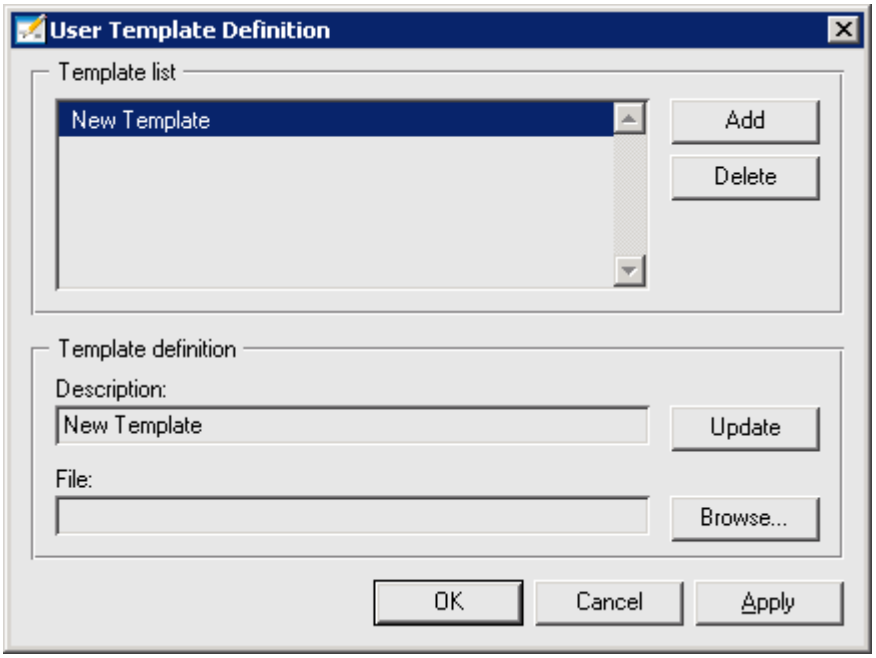


Figure 23: You can define the User Templates using this dialog box

3. In the File text box, type the name and path of the file to be used as a template or use the browse facility. Clicking **Browse...** opens the File Chooser dialog box for selecting the name and path of the template file. The description can also be changed. By changing and updating the descriptions the user can change the name shown when a User is chosen from the Template menu.
4. Click **Update**. If the description has been changed, the changes are made to the list at the top of the window.
5. Click **OK**.

In the File Chooser, the path for the file used as a template can be selected in four different modes. The modes are described in Section 6.3.3.

6.4 Object Handling

6.4.1 General

This section describes the basic rules on how to add, rename and select objects. Because the procedures are very similar in the Object List and in the object editors, they are described in the same section. All objects can be edited. This section also discusses the editing facilities, for example, cut, copy, paste, duplicate, resize, align and delete functions, as well as the View Object Tree function.

There are three kinds of objects: dialogs, images and dialog items. Images and dialogs are always added in the Object List. The dialog items, can be added either in the Object List or in the Dialog Object Editor. Normally, a dialog item is added in the object editor of a dialog (regular dialog or main dialog). The objects that are placed in the Object List are stored in the Visual SCIL Object definition (VSO) file and identified by their name. Hence, it is useful to add the dialog item in the Object List if the same item is to be loaded into several dialog systems.

Editing objects begins by selecting the object. All dialog items can be moved, copied, resized, and deleted. Choose basic editing functions from the **Edit** menus. There are also several other editing functions that can be chosen from other menus.

6.4.2 Testing an Object

Objects can be tested while they are edited. This way, the effect of definitions can be seen without closing the Dialog Editor. While testing a dialog the user can see how various features work. The resize feature, buttons and scroll bars can be tested, and text can be typed inside editable text fields. The feature is mainly used to test the user interface. To test an object:

1. Save the changes that have been made.
2. Select the object to be tested.
3. Activate the Dialog Editor Main Dialog.
4. Choose **Test Selected Object** from the Tools menu. A test copy of the object appears. The objects are activated for testing.
5. Test the object.
6. When the testing of the object is finished, choose **Stop Test** from the Tools menu.

6.4.3 Testing with Test Dialog

The Test Dialog can also be used for testing. The Test Dialog is opened by choosing **Test Dialog** from the Tools menu of the Dialog Editor Main Dialog. By choosing Test Dialogs, the functionality of the object can be tested. For example, the user can find out variable values and object properties. It can also be used for error debugging. The Test Dialog works the same way as when it is opened from the Tool Manager.

6.4.4 Undo and Redo

The Undo command cancels the most recent editing actions. Use it repeatedly to undo a series of actions. The Undo command identifies the most recent action that can be undone. The Redo command restores the most recent editing action canceled by the Undo command. Use it repeatedly to redo a series of actions. To undo or redo an action, choose the **Undo** or **Redo** command from the Edit menu.

6.4.5 Viewing Object Tree

Selecting **View Object Tree** opens a dialog, which contains a tree for all the objects and methods of the VSO file, and a read only text field for viewing the methods. The object rows contain images for the object type as well as the name and the type of the object. The methods are categorized into three categories by the color of the image:

- White: Public method
- Yellow: Private method, including cyclic methods and events
- Green: Predefined method, for example CREATE, INIT

Both the objects and the methods are sorted alphabetically, ignoring the text case. The object tree shows the objects as they were at the last saving of the VSO file. While performing a replace operation for an object in the **Object Tree** dialog box, that particular object should not be open at the same time in the object window shown in Figure 24. This is because the object in the object window is always updated when the window is closed, even if no changes were made, so any replacements made in the Object tree dialog box would be overwritten.

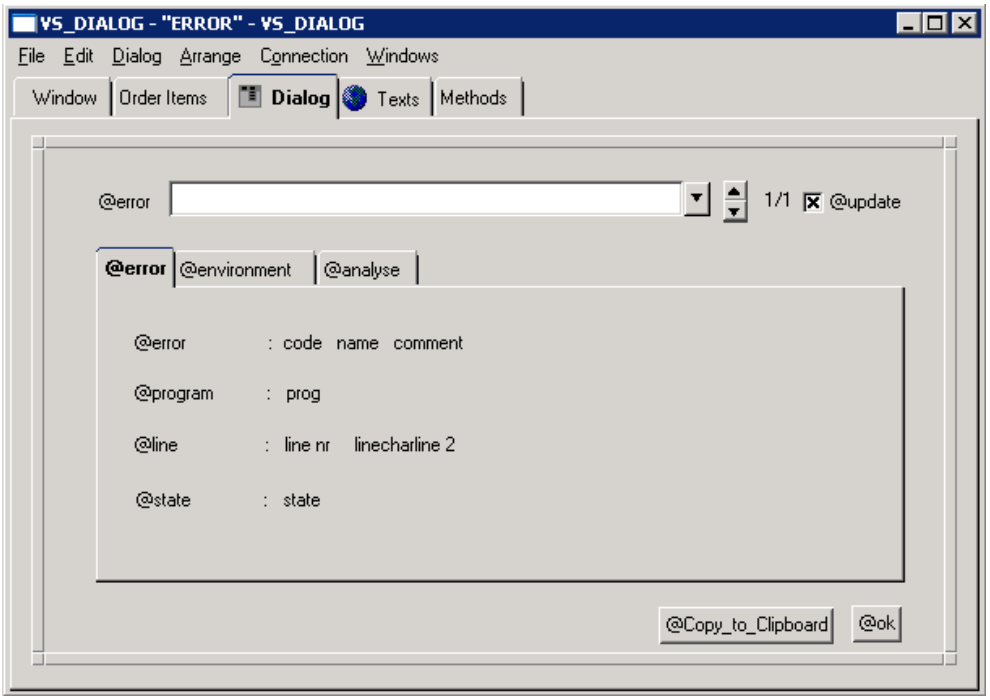


Figure 24: An Example of an object window

The Find/Replace functions work on the methods in the following way:

- The initial Find operation searches from all the methods of all the objects.
- The search direction is always from top to bottom.
- After successful Find operation the object tree is put into a filtered state, when only the objects and methods which contain the searched string are shown.
- If a row is selected from the object tree, while the filter is activated, then the next Find operation starts from that row.
- If a node is collapsed from the object tree, while the filter is activated, then the next Find operation applies only to the expanded nodes of the object tree.
- The Find/Replace operation is always started from the current cursor position, so moving the cursor either forwards or backwards inside a method is possible. If any text is selected, then the next scope starts at the second character of the selected text.
- When the Find/Replace dialog is closed, the object tree is reset and all the objects and methods are shown again.

6.4.6 Object list

6.4.6.1 Adding Objects

To add an object in the Object List:

1. Choose **New** from the Object menu. The list shown in Figure 25 appears.
2. Choose the type of the object you want to create.
3. The new object is shown in the Object List with a given name. The object has default attributes.

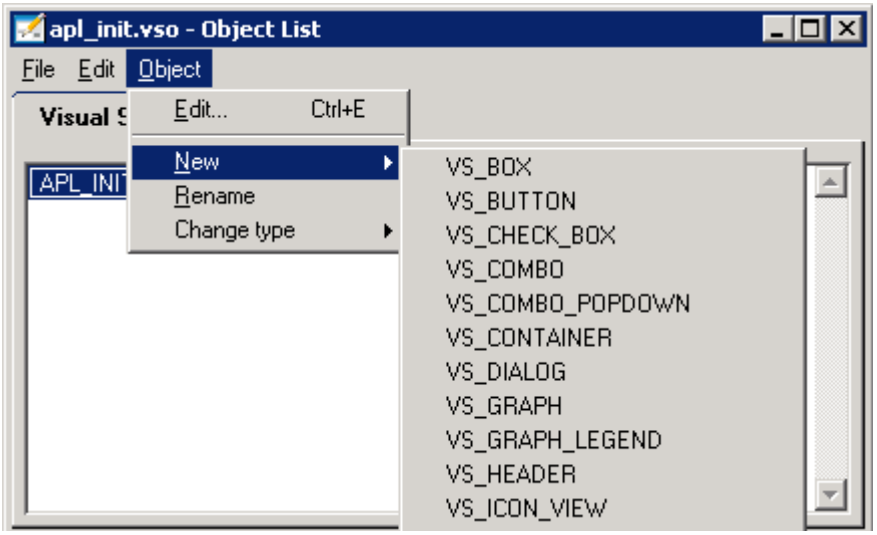


Figure 25: From the list choose the object type you want to add

6.4.6.2 Editing Objects

Objects are edited in their object editor. To open the object editor of the object on the list, double-click its name or select it and choose **Edit** from the Object menu. The Object Editor appears. To select an object that is listed on the Object List, click it. Editing is also possible for read only files, but the changes cannot be saved.

6.4.6.3 Renaming Objects

Objects that are listed in the Object List can be renamed. The names listed in it are the names used to identify the object within the file. They are also used in the SCIL command .LOAD when loading an object into a dialog system. To rename an object:

1. Select the object you want to rename.
2. Choose **Rename** from the Object menu. The dialog box shown in Figure 26 appears with the old name in it.
3. In the text box replace the old name with the new name. Click **OK**.
4. The new name of the object appears in the Object List.

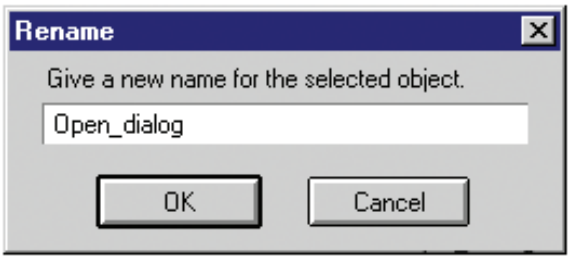


Figure 26: In this dialog box you type the new name for the object

6.4.6.4 Copying Objects

To copy an object within the Object List:

1. Select the desired object.
2. Choose **Duplicate** from the Edit menu. The new object appears.

Or:

1. Select the desired object.
2. Choose **Copy** from the Edit menu.
3. Choose **Paste** from the Edit menu. The new object appears.

The objects, which have been copied in the current object list, can be also pasted in a different Visual SCIL dialog editor instance running simultaneously with the first one or started later during the same monitor session.

6.4.6.5 Deleting Objects

The objects listed in the Object List are deleted in it. To delete an object:

1. Select the desired object.
2. Choose **Clear** from the Edit menu and the object disappears.

This can also be done by first selecting the desired object and then pressing DELETE or CTRL +X.

6.4.7 Object Editors

The editing procedure is similar regardless of whether the editor is opened from the Object List or from other object editors.

6.4.7.1 Adding Dialog Items

Dialog items can be added either in an object editor or in the Object List. Normally, they are added in an object editor because they are part of other objects. For more information on how to add them in the Object List, see Section 6.4.6.

To add a dialog item:

1. Open the Dialog or Main Dialog Object Editor into which objects should be added.
2. Choose **New Item** from the Dialog menu. The list shown in Figure 27 appears.
3. Click the dialog item type to be added.
4. The dialog item appears with handles and is now ready to be moved to its correct place.

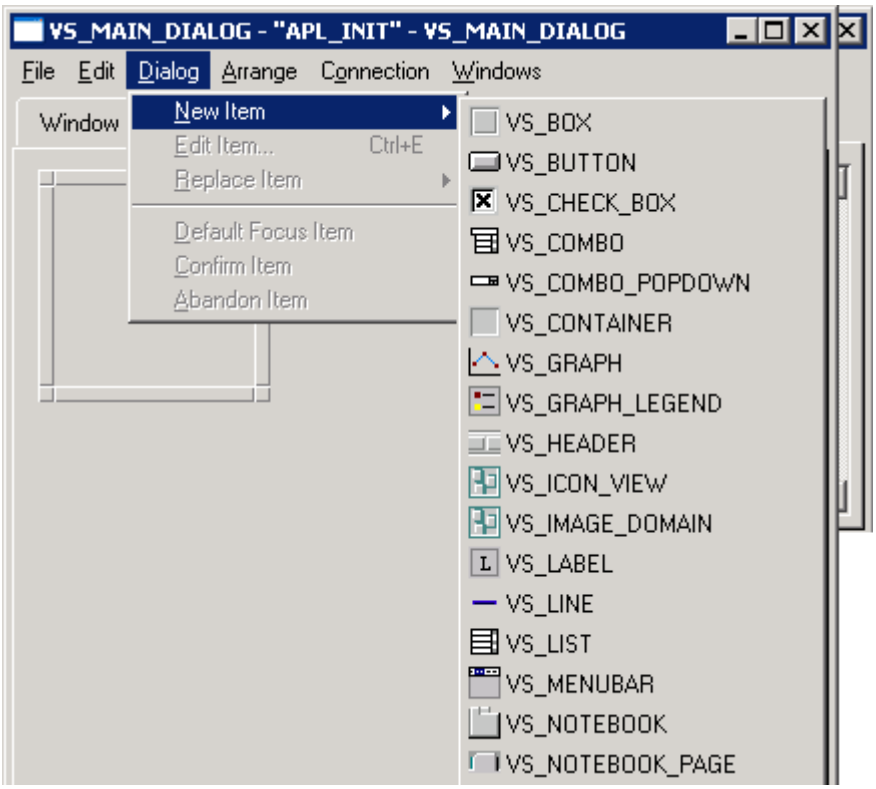


Figure 27: From the list shown above you can choose the type of dialog item you want to add.

6.4.7.2 Selecting Dialog Items

To select a dialog item in an object editor, click it. Handles appear on its border when the selection is successful.

Several objects are selected by holding the SHIFT key down while clicking on the objects to be selected. To select all dialog items, choose **Select All** from the Edit menu. This can also be done by pressing CTRL+A.

An item or several items can also be selected by selecting an area. To select an area, press the mouse button down in the corner of the area where the items to be selected are located. Hold the mouse button down while dragging the cursor to the opposite corner of the area. The items inside the area are selected.

6.4.7.3 Resizing Objects

To resize a dialog item:

1. Select the object.
2. Press the mouse button down on a handle and hold it down while dragging the cursor to the new position of the handle. The size of the dialog item is changed according to the new handle location.

To resize a dialog:

1. Activate the **Dialog** page of the Dialog Object Editor.
2. Hold the mouse button down on a border of the dialog while dragging the cursor to the new position of the border. The size of the dialog is changed.

6.4.7.4 Moving Dialog Items

To move a dialog item inside the parent object:

1. Select the dialog item.
2. Press the mouse button down in the middle of the item and hold it down while dragging the cursor to the new position.

The selected item can also be moved by using the arrow keys on the keyboard.

Dialog items can be moved from one object to another one. They can be placed into a dialog, main dialog, container or notebook. To move a dialog item from one object to another one:

1. Open the object editor for the parent object where the dialog item is currently placed.
2. Select the item.
3. Choose **Cut** from the Edit menu.
4. Open the editor for object where the dialog item should be added.
5. Choose **Paste** from the Edit menu. The item, which was moved from another object, appears.
6. Use the mouse cursor to move the item to the correct place.

6.4.7.5 Aligning Dialog Items

Several objects can be aligned according to their top, bottom, centre, left or right side. Alignment facilities place the dialog items so that one point of each object is on the same line.

- Top alignment arranges all the selected objects according to the uppermost selected object and its uppermost point.
- Bottom alignment places the objects according to the lowest selected object, and its lowest point.
- Center alignment is done either horizontally or vertically.
- Vertical centering moves the objects to the line that is located in the middle of the center points of the uppermost and lowest objects.
- Horizontal centers move the objects to the line that is located in the middle of the center points of the objects that are first in the left and last in the right side of the area.
- Left alignment places the objects according to a selected object that begins first in the left side of the area.
- Right alignment places the objects according the object that is selected and reaches the farthest point in the right side of the area.
- Distribute Horizontally places the dialog items evenly between the right and left items. Item positions do not change vertically.
- Distribute Vertically places the dialog items evenly between the top and bottom items. Item positions do not change horizontally.

To align:

1. Select the dialog items that should be aligned.
2. Choose the **Align** function from the menu of the object editor.

After the items have been aligned, their relative positions can be defined using connections.

6.4.7.6 Copying Dialog Items

To copy a dialog item in the object editor:

1. Select the Dialog Item to be copied.
2. Choose **Duplicate** from the Edit menu. The new dialog item appears.
3. Move the item to its correct position.

Or:

1. Select the Dialog Item to be copied.
2. Choose **Copy** from the Edit menu.
3. Choose **Paste** from the Edit menu. The new dialog item appears.
4. Move the item to its correct place. This is done by holding the left mouse button down while dragging the cursor to its new position.

The items that have been copied in the current item editor of the parent object can be pasted in the same or different object. The **Paste** operation can be fulfilled also in a different Visual SCIL dialog editor instance running simultaneously with the first one or started later during the same monitor session.

6.4.7.7 Changing the Dialog Item Layers

Dialog Items can be placed on top of each other, so that they are in different layers. Usually, dialog items should not overlap. The layering of dialog items specifies the tabbing order of the items when the TAB key is used to move from an item to another one. The layering/tabbing order can be viewed in the **Order Items** page in the Dialog, Container and Notebook Object Editors. For more information on the **Order Items** page, see Section 8.

The dialog item layers can be changed by using the **Bring To Front**, **Bring Forward**, **Send To Back** and **Send Backward** commands. The **Bring To Front** and **Bring Forward** commands move a dialog item forward in the dialog item layers. Choose **Bring Forward** to move a selected dialog item one layer closer to the front. **Bring To Front** moves the items to the front.

The **Send To Back** and **Send Backward** commands move a dialog item backward in the dialog item layers. **Send Backward** moves a selected dialog item one layer closer to the bottom. **Send To Back** moves the item to the back.

To change the dialog item layers:

1. Select the dialog item.
2. Choose one of the commands mentioned above from the menu.

6.4.7.8 Deleting Objects

To delete an object in an object editor:

1. Select the object to be deleted.
2. Choose **Clear** from the Edit menu and the object disappears.

Objects can also be deleted by first selecting the objects and then pressing DELETE or CTRL +X.

6.4.7.9 Replacing Objects

A dialog item can be replaced with another one by using the Replace function.

To replace a dialog item:

1. Select the dialog item to be replaced.
2. Choose **Replace** from the Edit menu. The list of dialog items appears.
3. Choose the dialog item type the first item should be replaced with. The dialog item appears in the dialog, in the same place as the previous item.

Section 7 Defining Objects

7.1 About this Section

This section describes the general principles of how to define VS objects.

Section 7.2	The first section describes the most common object definitions. These are the definitions that are given to almost all objects, for example, name and title, position, color and font.
Section 7.3	The second section describes the procedure for working with methods and SCIL Program Editor.
Section 7.4	The third section describes the procedure for working with language dependent texts.
Section 7.5	The fourth section describes geometry management. The relative location of objects is decided using connections. An explanation is given on how to define them.

7.2 Common Object Definitions

The most common definitions are shown in Figure 28.

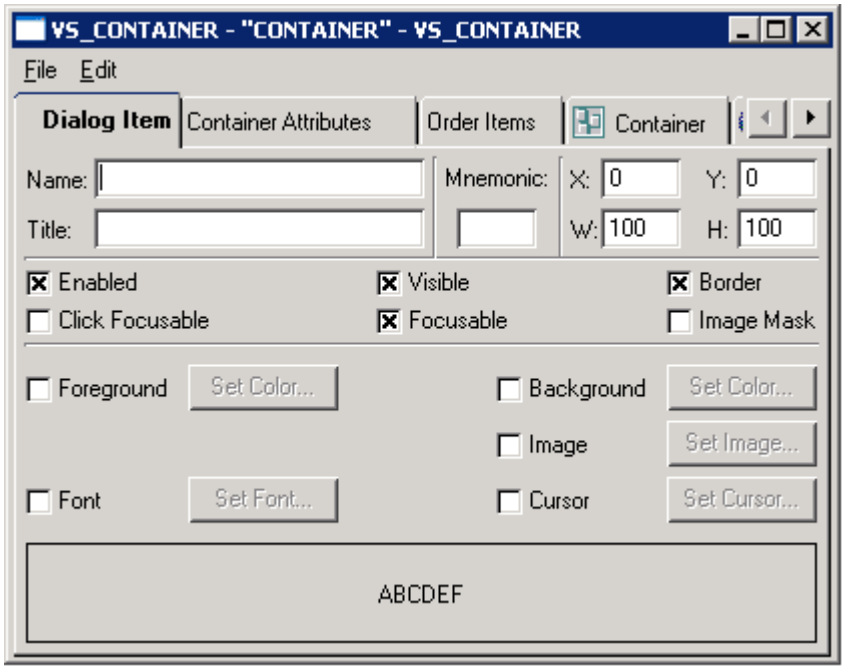


Figure 28: Some definitions are common for almost every object

7.2.1 Name

The Name is the object name of the dialog or dialog item, the descriptive name. It can contain 63 characters. In the Name text box, type the name to identify the object.

it is highly recommended to name every object, because unnamed objects cannot be accessed from SCIL or contain any methods. For objects listed in the Object List, the name given here has no meaning, because the name in the Object List will always be used.

It is useful to name objects with some predefined logic, otherwise it can be difficult to remember all the names when creating a large tool. Also, if objects added by the user are to be used by others, the logic of naming the objects should be explained somewhere. Note that it is not allowed to use the same name for two objects that have the same parent object. For more information on recommended logic for naming objects, see Section 5.

7.2.2 Title

The title is the text on an object, for example a button can have the text OK, or the title can be written at the top of a dialog. In order to be able to use the dialog with different languages, the language dependent text facility can be used. To enable translations, define the title as a text ID instead of a text. The text ID is the identification of the language dependent text (the text written after the character @). Translating language dependent texts is described in Section 7.4.

Example:

text ID in title @Yes

English: OK

Swedish: Ja

Finnish: Kyllä

German: Ja

7.2.3 Mnemonic

Use the Mnemonic field to specify a mnemonic key for a dialog item. A mnemonic key allows the user to choose the dialog item by pressing a specified key in combination with the mnemonic key modifier on the keyboard (usually the ALT key). Only one character can be specified as the mnemonic key. If the character is in the title, it will be shown underlined on screen. If text ID is used, the mnemonic key for text in every language is given in the translation table.

7.2.4 The Position and Size of Objects

The object's X and Y coordinates and the connections determine its position. For a dialog, these coordinates specify the distance from the lower left corner of the screen to the lower left corner of the dialog. For dialog items, the coordinates specify the distance from the lower left corner of the parent object to the lower left corner of the dialog item.

The size of the object is specified by width (W) and height (H) attributes. The default values can be freely edited. To edit them, click the text box and change the value. These values are also changed with the Resize and Move functions.

7.2.5 Color Setting

The foreground and background colors can be selected for all the objects. To select them:

1. Select the **Foreground** or **Background** option.
2. Click **Set Color**. The Color Chooser shown in Figure 29 appears.

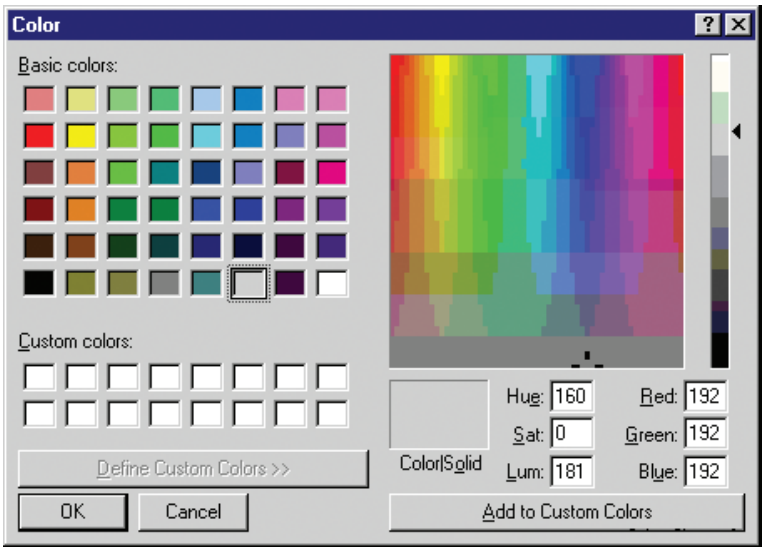


Figure 29: The colors can be changed in this tool

3. Choose the used Color Method from the **Color Method** drop-down list box at the top of the Color Chooser. HLS stands for a method where the user can choose and adjust the hue, light and saturation of a color using sliders. RGB stands for a method where the user can choose and adjust the shades of red, green and blue using sliders. CMY stands for a method where the user can choose and adjust shades of cyan, magenta and yellow. Gray stands for a method where shades of gray are chosen.
4. Make sure the **Color Wheel**/**Color Plane** option is selected from the View menu.
5. Click the color that should be used in the color view in the upper right corner of the Color Chooser. Then, adjust it using the sliders. The color box in the upper left corner of the Color Chooser displays the color selections. The top half of the color box displays the previous selection and the bottom half the current selection. The current selection can be changed to the previous selection by clicking the top half of the Color Box.
6. Click **OK** to apply the color and close the Color Chooser or **Apply** to apply the color and let the Color Chooser stay open.

From the **View** menu the user can also choose to view Named Colors and Color Tolerances. Named Colors displays a list of predefined colors. Select colors for the application directly from the list. The Color Tolerances displays a set of tools used to define the appearance of colors on screen. Define the appearance of colors by specifying a method for dithering.

Use the saved color bar to save frequently used colors. To save a color:

1. Choose a color.
2. Hold the CTRL key down while clicking a box in the bar. The color is added to the chosen box.

7.2.6 Font Setting

The font type can be selected for all objects:

1. Select the **Font** option.
2. Click **Set Font**. The Font Chooser shown in Figure 30 appears.

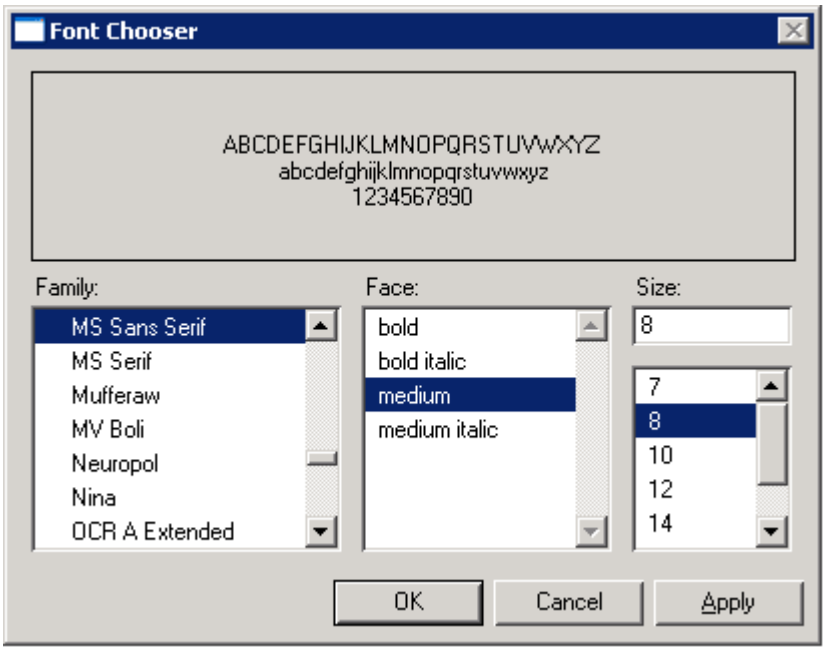


Figure 30: In the Font Chooser you can choose the font type you want to use

3. Select the desired font family in the **Family** list on the left. The font types included in it are listed in the **Face** list.
4. Click a Face option and then select or type in the size of the font in the **Size** box. A sample of the font is shown in the upper part of the Font Chooser.
5. Click **OK**.

7.2.7 Cursor

The cursor is the appearance of the cursor (mouse pointer) when the cursor is placed on an object. To define the appearance of the cursor:

1. Select the **Cursor** option in an object editor.
2. Click **Set Cursor**. The Image Editor appears.
3. Specify the appearance of the cursor. Working with the Image Editor is explained in detail in Section 10.

7.2.8 Icon

An icon is shown in the icon bar when a dialog is minimized. Hence an icon is an image that can be defined for dialogs. The icon of a dialog is shown only on a local VS monitor. To define an icon:

1. Select the **Cursor** or **Icon** option in an object editor.
2. Click **Set Cursor** or **Set Icon**. The Image Editor appears.
3. Define the appearance. Working with the Image Editor is explained in detail in Section 10.

7.2.9 Image

Boxes, buttons and containers may contain an image. If a button contains an image, it is shown on the button. If a box or a container contains an image, the image is adjusted to fill the whole object.

To define the appearance of the image:

1. Add the image object in the Object List. Double-click it to edit.
2. Define the appearance. Working with the Image Editor is explained in detail in Section 10.

An image created this way must be loaded at runtime.

Example:

```
.load root\myimage=vs_image("file.vso", "image")
.set root\btn._image="root\myimage"
```

Another way to assign a static image to an object is to check the Image check box in the first page of the object editor and define the image there.

7.2.10 Other Attributes

The following attributes are described in more details in SYS600 Visual SCIL Objects where all attributes are listed. Hence, the following list only contains the references to it. See the attribute in the description of the object that is being defined. If the attribute is not described in it, it does not have effect on the object.

- Enabled. The option is selected by default. For more information see the attribute _ENABLED in SYS600 Visual SCIL Objects.
- Visible. The option is selected by default. For more information see the attribute _VISIBLE in SYS600 Visual SCIL Objects.
- Border. The option is selected by default. For more information, see the attribute _HAS_BORDER in SYS600 Visual SCIL Objects.
- Click Focusable For more information, see the attribute _CLICK_FOCUSABLE in SYS600 Visual SCIL Objects.
- Focusable. For more information, see the attribute _FOCUSABLE in SYS600 Visual SCIL Objects.
- Image Mask. For more information, see the attribute _HAS_IMAGE_MASK in SYS600 Visual SCIL Objects.

7.3 Methods

All VS objects can have methods. Methods, which are SCIL programs that define the behavior of the object, are defined in the **Methods** page of an object editor. In the upper part of the editor are five methods: create, init, delete, error and help methods. The lower part of the editor show property pages for public, private, action, cyclic and event methods. For more information, see SYS600 Visual SCIL Objects and SYS600 Programming Language SCIL.

7.3.1 Handling Methods

7.3.1.1 Defining Methods

To define a method:

1. Select the method and click **Edit**. The methods that are of type public, private, action, cyclic or event are located in the property pages. If the method does not exist, the user can add it. To add a method, click New and a new method of the type appears.
2. The SCIL Program Editor appears and the existing program is shown. In the editor the user can copy, move or delete selections. It is also possible to insert a line or a comment mark, and use update, find, replace, undo and redo features. The features of the SCIL Program Editor are described in more detail in Section 7.3.1.4.

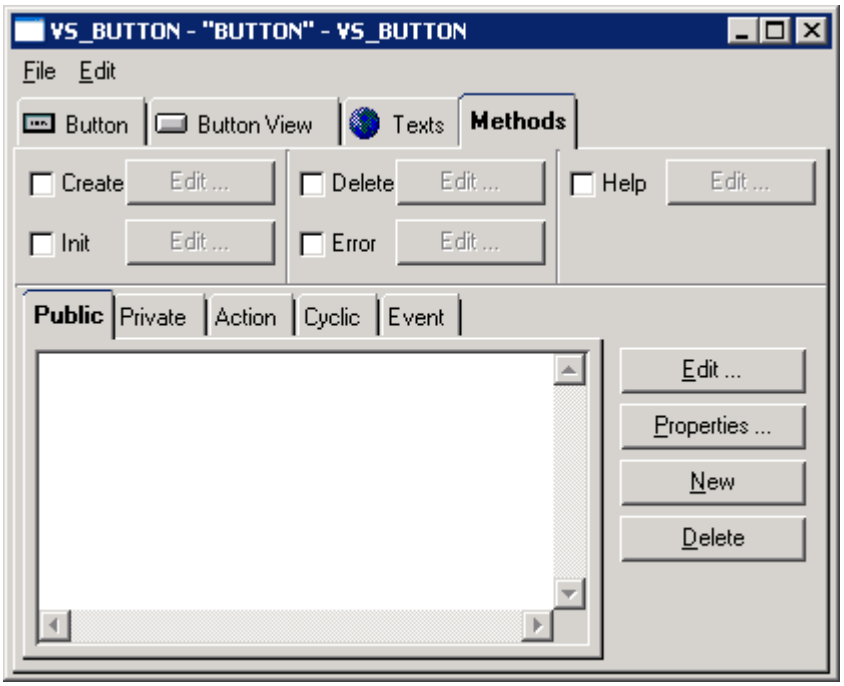


Figure 31: The procedure of defining methods is started from the Methods page of the Object Editor.

3. Close the SCIL Program Editor by choosing **Exit** from the File menu.

7.3.1.2 Adding New Methods

The user can add new methods of type public, private, cyclic and event. Action methods are predefined in the system and cannot be added, renamed or deleted. They can, however, be enabled or disabled. To add a new method:

1. In the object window, click the tab with the right method type name.
2. Click **New**. The dialog box shown in Figure 32 appears with the default name NEW_METHOD.

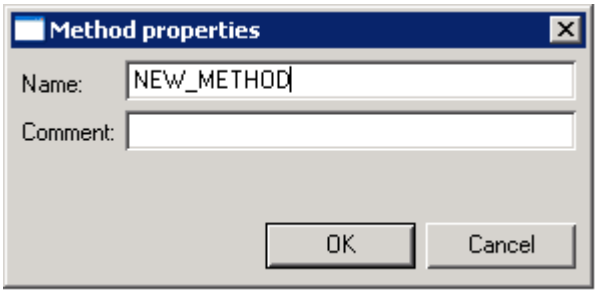


Figure 32: The name of the method can be changed in this dialog.

3. Replace the default name with the name of the method in the **Name** text box. A comment can also be typed in the Comment text box.
4. Click **OK**.

Renaming of a method can be done by selecting the method in the object window, clicking **Properties** and performing similar steps as the steps 3 and 4 for adding a new method.

7.3.1.3 Deleting Methods

The user can delete methods of type public, private, cyclic and event. To delete a method:

1. Select the method to be deleted.
2. Click **Delete**.

The program related to an action method can also be deleted. To delete an action method:

1. Clear the check box of the method to be deleted.
2. The question "Do you really want to delete the selected action method?" is shown. Click **Yes** to delete.

7.3.1.4 Using the SCIL Program Editor

The File menu of SCIL Program Editor contains options for file handling, updating and exiting the editor. The Edit menu contains options for copying, moving and deleting. The **Undo** option found here is useful when reverting to previous operation. The Search menu contains options, which help the programmer search and replace text strings. There is also a **Go To Line** option for moving to a certain line. In the Settings menu, the font size can be changed. The Help menu provides information about the editor.

The SCIL Program Editor is operated according to the same principles as other Windows based applications. The editor is specialized for SCIL program editing. For more information on using the SCIL Program Editor, see SYS600 Programming Language SCIL.

7.4 Language Dependent Texts

By using the Dialog Editor, the user can add and translate language dependent texts into several languages. The title attribute, as well as arbitrary texts, can be changed into several languages. This is done in the **Text** page of an object editor. For example, Notice Dialogs and texts in the SCIL Program Editor can be translated into local languages. The text to be translated is found in the stdlang.vso file, in ../sc/prog/exec -folder.

It is recommended that language dependent texts are not translated separately for every object, the translations could be collected somewhere. For example, translations could be inserted in the **Text** page of the main dialog. A text can also be translated dynamically by using the TRANSLATE function. For more information on this, see the first part of this manual or SYS600 Programming Language SCIL.

7.4.1 Inserting New Languages

When a new object is added, it has, by default, one language (English) inserted in it. The languages into which the translations are done should be inserted first.

Insert new languages by choosing **New Language** from the Text menu. Type in the language identification to be used and then click **OK**. The language identification should follow the ISO 639 standard.

7.4.2 Inserting Text IDs with Translations

In the object editor, two different languages can be displayed at the same time. The procedure for inserting Text IDs with translations into two languages is described here. To insert more languages, repeat the procedure.

1. The identifications of the inserted languages are shown in the **Language 1** drop-down list box. Choose one.
2. The identifications of the inserted languages are shown in the **Language 2** drop-down list box. Choose one, but note that it has to be different from the first language.
3. From the Text menu, choose **New Text ID**. Type the text ID in the first column. The text ID is the identification of the language dependent text, that is, the text written after the character @, for example in title. Then click **OK**.
4. In the second column, type the text in the language chosen to be Language 1. To define a mnemonic key, insert an ampersand character (&) before the character that will be underlined. In the third column, type the text in the second language.

7.4.3 Renaming Text IDs

Text IDs can be renamed. To rename a Text ID:

1. Click the desired Text ID.
2. Choose **Rename Text ID** from the Text menu.
3. Type the new name for the Text ID.
4. Click **OK**.

7.4.4 Deleting Text IDs

To delete a Text ID:

1. Click the desired Text ID.
2. Choose **Delete Text ID** from the Text menu.
3. Click **Yes** to delete the Text ID. The Text ID is deleted.

7.5 Geometry Management

7.5.1 General

The distance between different objects in various situations and the size of some dialog items are defined by using connections. Dialog items have both inside and outside connections. Inside connections determine the size of an item. Outside connections determine the relative location of a dialog item with the border of the dialog or another dialog item. This feature is needed particularly when resizing a dialog containing dialog items.

There are seven types of Connections: Natural Length, Natural Base + Spring, Fixed Length, Fixed Base + Spring, Locked Fixed Length, Zero Length and Spring. These names are used when connecting one object to another. The procedure for using the Connection Editor is described on Section 7.5.3. The size and place of objects may be changed after they have been added. If the place of one object changes, the places of other objects may also have to be changed. If connections are used, the user can decide which distances will change and which ones will always be same. The connections for dialog items are defined in the object editor of the parent object.

When defining connections, it is recommended to have **Use Large Connections** and **Show** selected on the Connections menu.

7.5.2 Connection Colors

Connections are shown with different colors that have specific meanings. The meanings of colors are listed below.

- Blue: The connections are colored blue by default.
- Green: The part of a connection that is colored green is of type Natural Length.
- Yellow: The connection that is colored yellow is the one that is selected. The type of this connection can be changed.
- Red: If connections are colored red, there is an error relating to them. For example, if the connections from an object to the left border of the dialog and to the right border of the dialog are both fixed, their color is red because there is an error. There must always be one horizontal and one vertical flexible connection. An object should never be left with illegal connections.

7.5.3 Connection Types between Objects

- Natural Length: The distance will always be 5 points.
- Natural Base + Spring: The distance will be always be at least 5 points. If one object is moved farther away, the distance will grow.
- Fixed Length: The distance will be the same as it was at the moment of choosing a fixed connection. The distance can be changed by stretching it with the mouse or by changing the Value text box in the Connection Editor. It will not be changed automatically when the places of other objects are changed.
- Fixed Base + Spring: The base distance will be at least as long as it was at the moment of choosing this connection. Hence, the objects cannot be moved closer to each other, but they can be moved farther away from each other. If the user is changing the connection type from Spring to Fixed Base + Spring, the fixed base is initially zero. The base distance can only be changed in the Value text box in the Connection Editor.
- Locked Fixed Length: The distance will always be the same as it was at the moment of choosing a fixed connection. The distance can be changed only by changing the connection type or the Value text box in the Connection Editor.
- Zero Length: An object will begin at the point where the previous one ends, there will be no distance between them.
- Spring: The distance will change freely when the objects are moved.

7.5.4 Examples for connection type between objects

7.5.5 Geometry Management

Geometry management in Visual SCIL user interface elements is based on springs and struts defined by using the Connection Editor in Dialog Editor. Connections are determined by using different types of springs and struts.

7.5.6 Natural Length

When Natural Length is used inside dialog item, the object adjusts automatically to changes according to font and text size. The width and height of the object is always the smallest possible. When Natural Length is used between objects, it forces the distance between adjacent dialog items to 5 points.

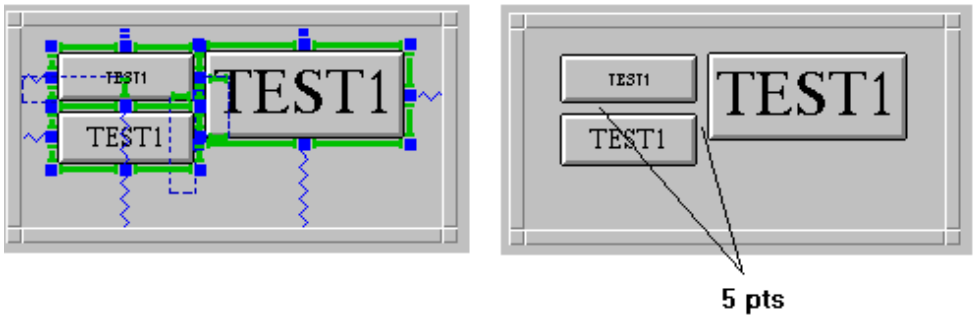


Figure 33: The connections inside and between buttons have been done with natural length.

7.5.7 Spring

When Spring is used inside a dialog item, the object enables changes in object sizes and distances between adjacent objects.

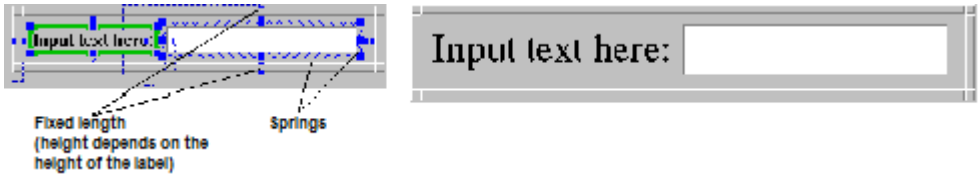


Figure 34: Horizontal and vertical springs in the text field enable the size to change depending on the label-size in the left.

7.5.8 Natural Base Added with Spring

Similar to the Natural Length, objects defined by using this type of connection are adjusted automatically to changes according to font and text size. The difference to Natural Length is that the dialog item is adjusted according to changes in dialog size, too. For example, the object stretches when the dialog is resized. In Figure 35, connections in dialog items have been internally defined as natural base added with spring in horizontal direction. Adjacent objects have been defined by using the connection type natural length.

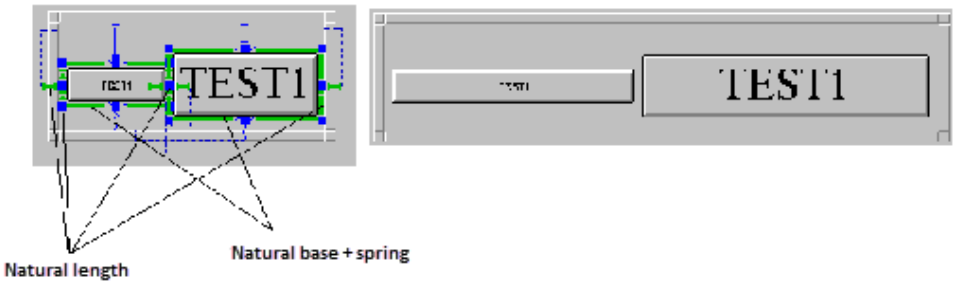


Figure 35: The connections have been done by using natural base added with spring and natural length.

7.5.9 Fixed Length and Locked Fixed Length

This connection type defines the size of an object or the distance between adjacent objects to a certain fixed value. The difference between Fixed and Locked Fixed Length type is that the first one enables the dragging of dialog item in Connection Editor, but the latter does not. Fixed Length is not adjusted to changes according to font and text sizes. E.g. the label defined as type Fixed Length cuts off those pixels that do not fit inside label area. Connections of Fixed Length type can be found by setting some large font, like ("M", 4) to the dialog. Note how the width of the left button in Figure 36 is not adjusted according to the dialog resize operation due to the connection type Fixed Length.

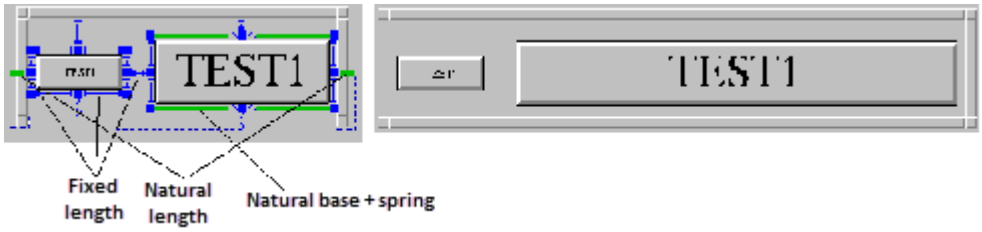


Figure 36: Connections between adjacent objects and inside objects have been done by using fixed length type.

7.5.10 Fixed Base Added with Spring

Specifies a minimum base length added with a spring. When used inside dialog item, the width or height of the object is always same as or greater than Fixed Base. The spring enables the stretching of the object depending on dialog size. When used between the objects, Fixed Base guarantees a certain minimum fixed base distance between objects. Due to the spring, the distance can be larger than the specified value, but never smaller than this base distance.

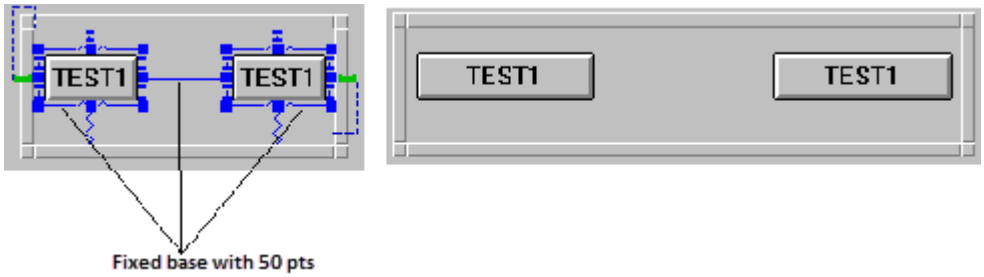


Figure 37: Connections inside and between buttons have been done by using fixed base value 50 points and spring. Due to this, the width of the objects and distance between them, can not be under 50 points.

7.5.11 Changing the Connection Types between Objects

The connections between a dialog item and the dialog, or between dialog items themselves, are discussed below. When an item is initially placed in a dialog, the outside connections on the top and left sides of the item are fixed by default. To clear all the changes made to the connections, choose **Reset** from the Connection menu. To change the connection type:

1. Select the option **Show** from the Connection menu.
2. Select the dialog item.
3. Select the connection whose type is to be changed.
4. Choose the connection type to be used from the **Connection** menu.

To connect dialog items to other dialog items, use the Chain feature. A chain is a series of connections between dialog items. A chain is in one direction, horizontal or vertical, and must always include at least one spring connection. If there are no spring connections, the chain cannot stretch to fit a resized dialog. To use the chain feature:

1. Select **Show** from the Connection menu.
2. Select the dialog items.
3. Choose **Chain** from the Connections menu. The submenu shown in Figure 38 appears. From the submenu, choose the desired connection type. The figures before the different connection types help the user to choose the correct one.

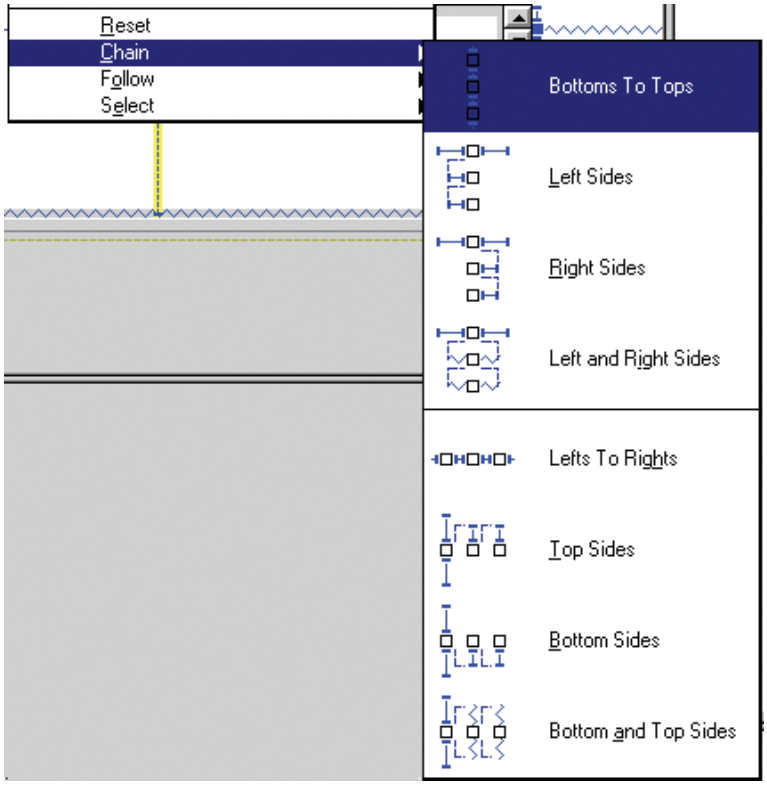


Figure 38: The Chain submenu

7.5.12 Example of Connecting Buttons

This is an example of connecting two buttons to a dialog. The buttons are connected to each other with the chain function Lefts to Rights so that the connections are fixed to the objects on their left side. See Figure 39. Also, the Chain connection function Bottom and Top Sides is used to connect the top and bottom sides of buttons with fixed connection to each other. Then, the **OK** button is connected with a fixed connection to the bottom side of the dialog. When the dialog is enlarged, the distance between the buttons and the left and bottom sides of the dialog stay the same. See Figure 40.

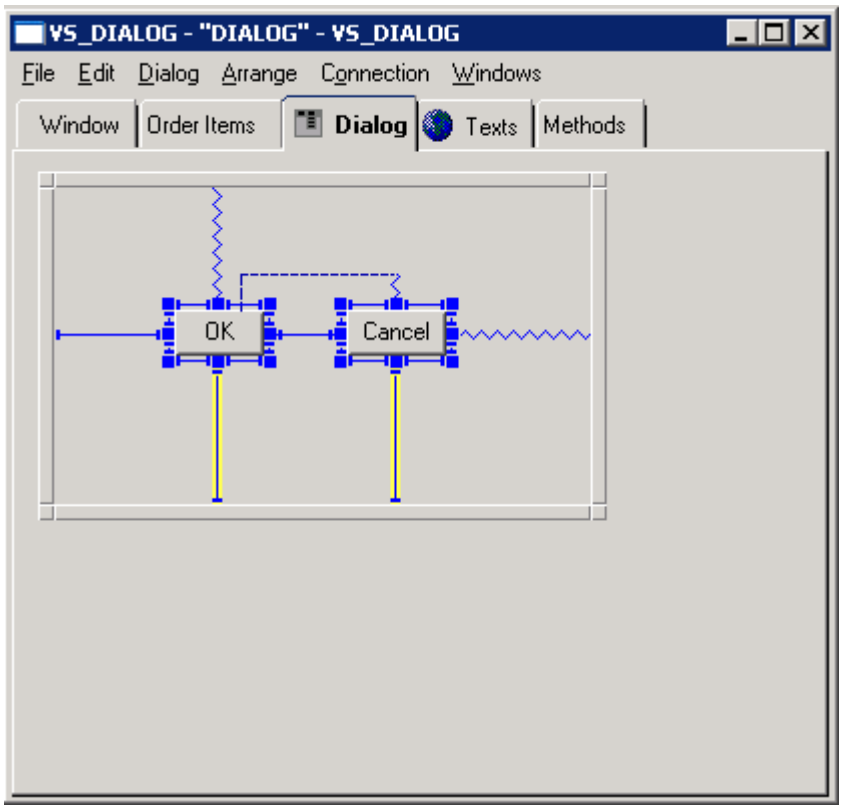


Figure 39: Button connections are made with the Chain connection function. The connections to the left and bottom sides of the dialog are fixed.

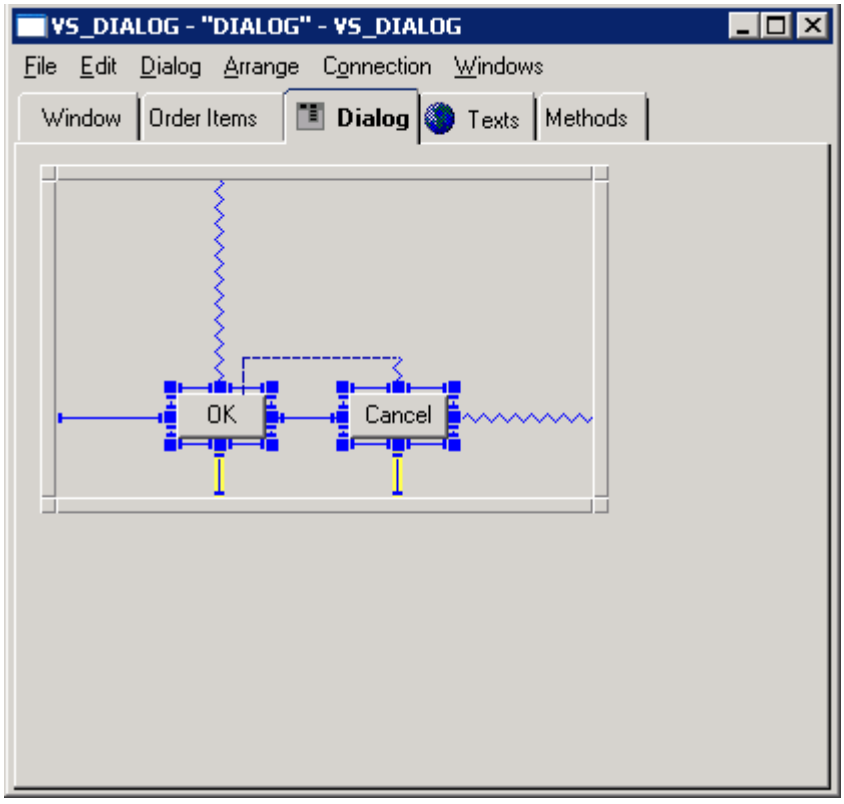


Figure 40: When you enlarge the dialog, the distance between the buttons and left and bottom sides of the dialog remains the same.

7.5.13 Connection Types inside an Object

The inside connections of a dialog item are contained within the dialog item, connecting opposite sides of the item. They determine its size.

- Natural Base: Distance is object dependent. The size of the object is flexible. For example, when the font size is changed, the size of a button or label is also changed. The size is adjusted to fit the font size and text length so that it will require the smallest possible space. The distance can be changed while editing the connections.
- Natural Base and Stretch: The object can be enlarged freely. The minimum distance is object dependent.
- Natural Base and Drag Lock: The connection cannot be changed without removing the lock. The minimum distance is object dependent.
- Fixed Base: The distance will be the same as it was at the moment of choosing a fixed connection. The distance can be changed by stretching it with the mouse or in the Value text box, but it will not be changed automatically.
- Fixed Base and Stretch: Specifies the minimum length of the distance. It will be at least as long as it was at the moment of choosing this connection. Hence, the sides of objects cannot be moved closer to each other without changing the connection type, but they can be moved farther away. If the user is changing the connection type from Stretch to Fixed Base and Stretch, the fixed base is initially zero. This minimum length can only be changed in the Value text box.
- Fixed Base and Drag Lock: The distance will always be the same as it was at the moment of choosing a fixed connection. The distance cannot be changed without changing the Value text box or changing the connection type.

7.5.14 Changing the Connection Types inside an Object

The connection types inside an object can be changed in the same way as the connections between objects, or in the way that is described here. To change the connection:

1. Select the object.
2. Open the Connection Editor by choosing **Connections** on the Window menu.
3. Select the connection whose type is to be changed. At the top of the editor the name of the object and where the connection points are can be seen. See Figure 41.
4. Select **Natural Base** or **Fixed Base**. If **Fixed Base** is selected, also enter the value, the length of distance in a number of points.
5. **Stretch** or **Drag Lock** can also be selected.
6. Close the Connection Editor by clicking **Close**.

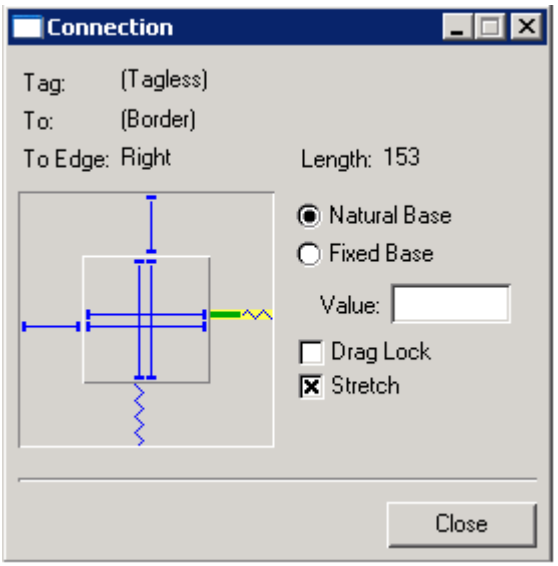


Figure 41: The Connection Editor

7.5.15 Source and Destination of a Connection

Connections have a direction determined by the source and destination of a connection. The source for outside connections is always a dialog item and the destination is another dialog item or border of a dialog. The source for inside connections is one edge and the destination is the opposite edge of the item. The destination of a connection is usually changed when the object the item is connected to is changed. So actually the direction of a connection does not define how the connection behaves, only the objects connected to each other. The connection type between them defines the behavior.

The destination of a selected connection can be seen in the **TO** text field in the upper left corner of the Connection Editor. If the destination is another dialog item, the name of the item can be seen. If the destination is the border of the dialog, the word Border is displayed. If the destination is inside the dialog item, the word Self is displayed. The destination of a connection can be changed by selecting a connection and dragging it to its destination.

The **To Edge** text field shows the location of the connection on the destination item. The location is expressed as the top, bottom, left or right edge of the destination item.

Usually, the destination of a dialog item can be seen in the picture by selecting it and following the connection. The object at the other end of it is the one it is connected to. If, for some reason, the destination cannot be seen properly, the destination of a connection can also be found by using the Follow function. This is done by selecting a dialog item and a connection and then choosing **Follow** from the Connections menu. Choose the connection side from the submenu. If the item is connected to another item on that side, the object to which the item is connected to is selected.

Section 8 Container Group Objects

8.1 About this Section

This section describes the general features related to defining container group objects. Container group objects are the Visual SCIL objects that can contain other objects.

Some definitions and functions are common to most of the objects. These are discussed in Section 6. This section discusses all the other functions related to defining dialogs, notebooks, containers, picture containers and menus.

Section 8.4	The first section describes how to define dialogs.
Section 8.5	The second section describes how to define notebooks.
Section 8.6	The third section describes how to define containers.
Section 8.7	The fourth section describes the general facts related to menus, how to define menu bars, menus and menu items.
Section 8.8	The fifth section describes how to define picture containers.

8.2 General

The procedure for adding objects and the basic editing facilities are described in Section 6. The most common definitions, name and title, color and font setting, cursor, image, enabled, visible, border, click focusable, focusable and image mask are also described there.

Notebooks, containers, menus and picture containers are dialog items, so they can be inserted into other objects. Dialogs, notebooks and containers can contain all types of dialog items. There is no limit to how many levels of objects can be placed inside each other.

Notebooks, containers, menus and picture containers are added either in the Object List or in an object editor. Dialogs can be added only in the Object List.

8.3 The Order Items Page

The **Order Items** page is common to dialogs, notebooks and containers, which is why it is discussed here. The dialog items inserted in an object are listed in the **Order Items** page. They are in the same order in which items are selected when the TAB key is pressed.

The order of the items can be changed by selecting the row and holding down the middle mouse button or both left and right buttons while dragging the row to a new position. The list includes the dialog item type, its name and title.

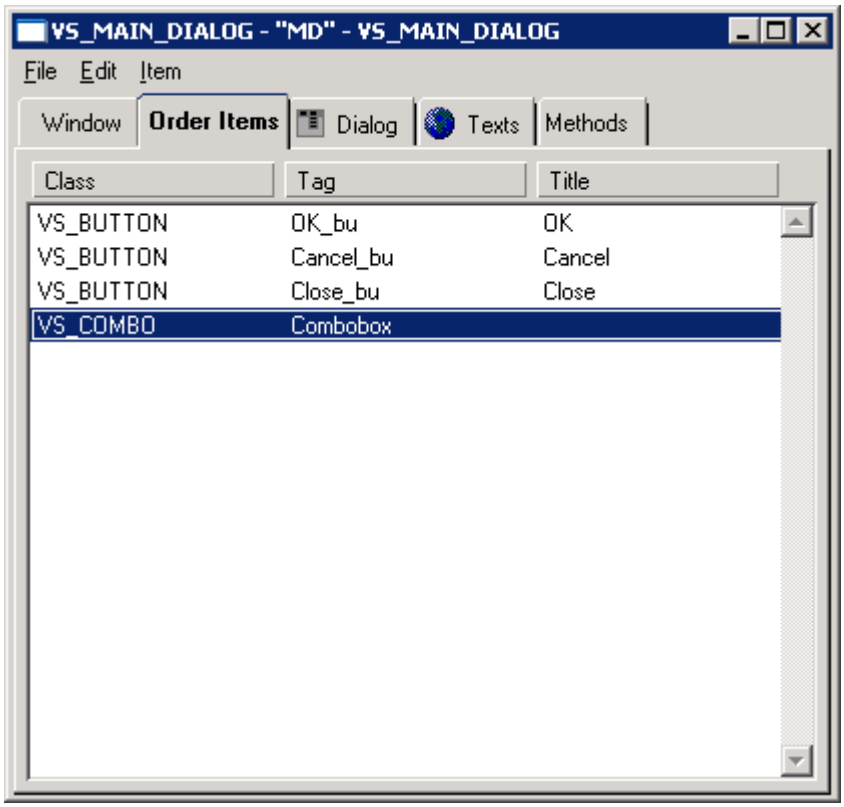


Figure 42: The Order Items page in which the dialog items inserted in the dialog are listed

In the **Item** menu, there are options that specify the behavior of items in the dialog. The options are **Default Focus Item**, **Confirm Item** and **Abandon Item**. The dialog item selected when the **Default Focus Item** is chosen is the item that will have the keyboard focus when the user opens the dialog.

Choosing the **Confirm Item** option for a dialog item means that if the user presses ENTER, the item behaves as if it was clicked. There can be only one confirm item in a dialog. Choosing the **Abandon Item** option for a dialog item means that if the user presses ESC on the keyboard, the item behaves as if it was clicked. There can be only one abandon item in a dialog.

8.4 Dialogs

8.4.1 General

Dialogs are used when dialog boxes, which are windows appearing independently on the screen, are desired. There are two types of dialogs: main dialogs and dialogs. Usually, it is recommended to use main dialogs. The types differ in the way variables can be read from other dialogs.

Because the working procedure for both of them is the same, they are discussed together. Dialog items are placed in a dialog. The position of a dialog item is specified with connections.

Dialogs are edited in the Dialog Object Editor, which contains five pages. The **Window** page and the **Order Items** page are described in this section.

8.4.2 Window Page

The first page of the Dialog Object Editor is the **Window** page. It contains several attributes that are already explained in Section 6. The attributes discussed in this section are the attributes located at the bottom. See Figure 43.

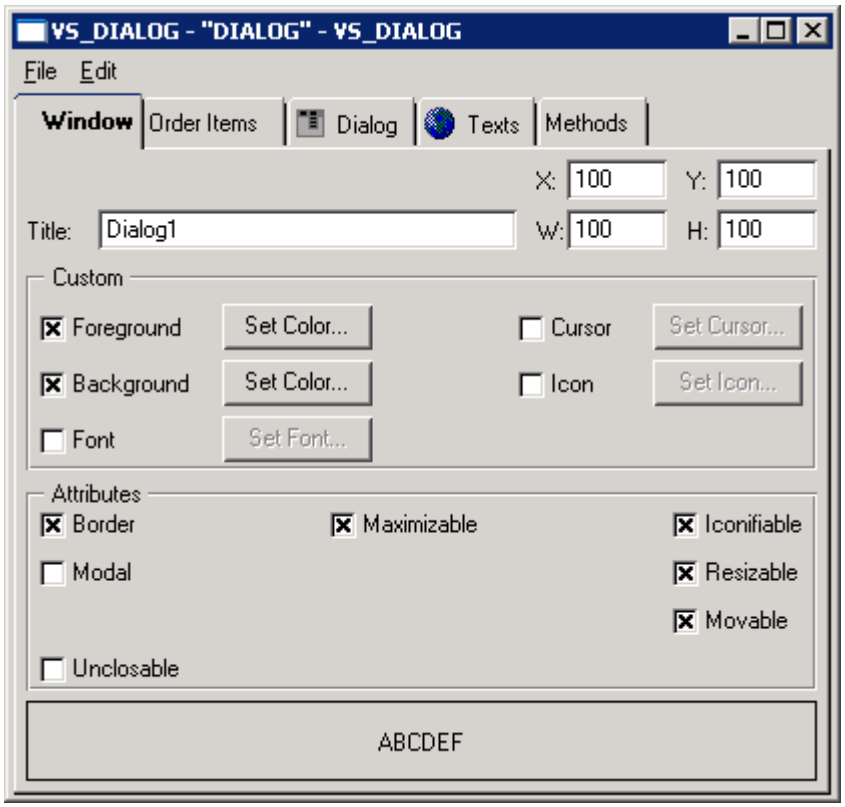


Figure 43: The Window page of the Object Editor when defining dialogs

- **Border** is selected by default. The dialog is outlined with a border.
- **Maximizable** is selected by default. The dialog can be enlarged to fill the whole screen.
- **Iconifiable** is selected by default. The dialog can be minimized to an icon.
- **Modal** is not selected by default. When it is selected, the dialog box is the only active window in the application. The user must respond or perform some actions in the dialog to change the focus.
- **Resizable** is selected by default. The user can resize the dialog.
- **Movable** is selected by default. When it is selected, the user can move the dialog on screen.
- **Unclosable** is not selected by default. The user cannot close the dialog by clicking the **Close** button (the **X** button) in the title bar of the dialog.

For more information on the attributes, see the _STYLE attribute in SYS600 Visual SCIL Objects.

8.5 Notebooks

8.5.1 General

Notebooks are often used to group dialog items that are somehow related to each other. Usually, there are many dialog items in a notebook, a there would not be enough space for them in a container. See Figure 44. For example, a notebook can contain several check boxes

and option buttons. A notebook usually contains several pages, which all have a tab to enter the page. Dialog items placed in notebook pages are edited in their own object editors.

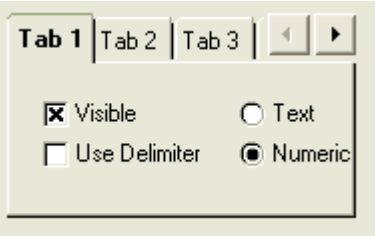


Figure 44: An example of a notebook

Tabs are managed and displayed by tab sets. A tab set contains all the tabs for pages and sections at the top level of a notebook. The name of a page is shown on its tab.

Various attributes of a notebook determine its behavior and appearance. These attributes can be defined in the Custom Style Area of the **Notebook** page.

There are six pages in the Notebook Object Editor. The common attributes of the first page are discussed in Section 7. Most definitions are performed in the **Notebook** page. These functions are described in Section 6.

Inserting and translating the language dependent texts is done in the **Texts** page. Adding and editing methods is done in the **Methods** page. The functions on these pages are described in Section 7.

The only page that differs from pages in other object editors is the **Notebook** page where the user can add the notebook pages.

8.5.2 Defining a Notebook

To define a notebook:

1. Enter the **Notebook** page in Notebook Object Editor.
2. To add a notebook page, choose **New Page** from the Notebook menu. See Figure 45.

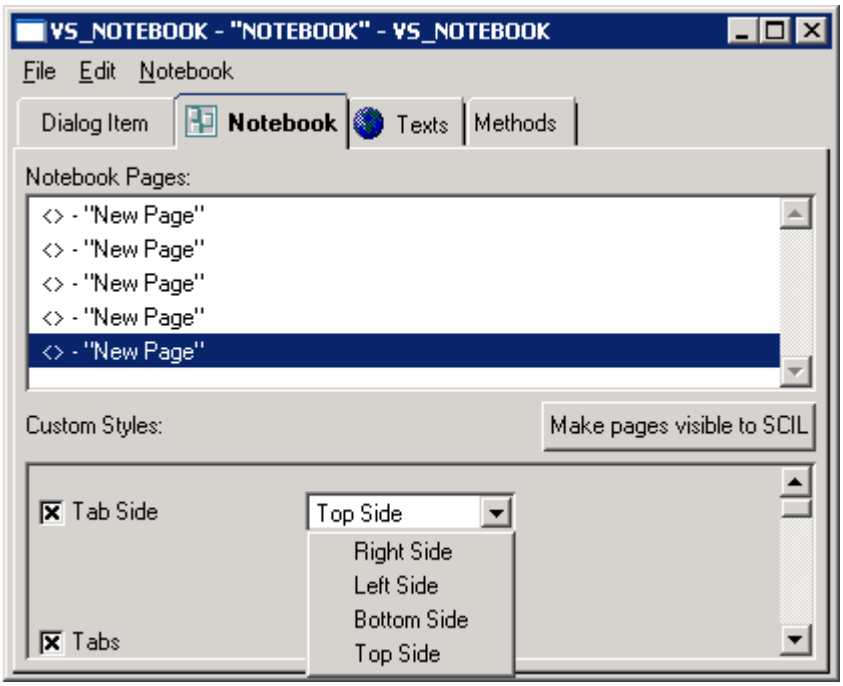


Figure 45: The Notebook Object Editor

3. Open the Notebook Page Object Editor by double-clicking the new page, the notebook page. The Notebook Page Object Editor is shown in Figure 46.
4. To add a new dialog item to the notebook page, choose **New Item** from the Page menu.

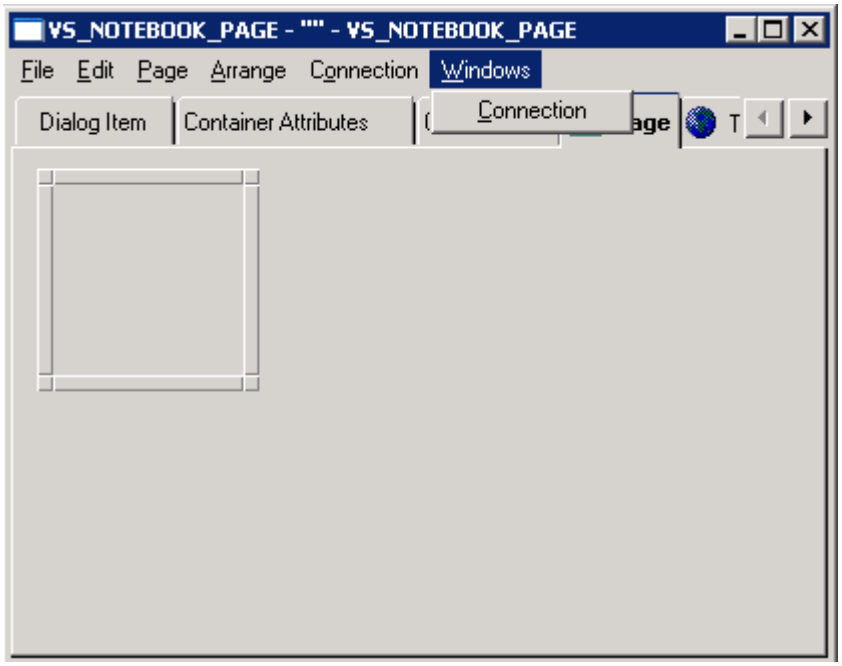


Figure 46: The Notebook Page Object Editor

The **Order Items** page contains a list of dialog items that are inserted in the **Notebook** page. For more information on defining this page, see Section 8.3.

5. Close the Notebook Page Object Editor by choosing **Close Editor** from the File menu.
6. To customize the appearance of the tabs, use the options in the Custom Style area at the bottom of the Notebook Object Editor. The attributes in the Custom Style area are listed later (The Custom Style Options) with an explanation.
7. Click the button with the text **Make pages visible to SCIL**, otherwise the pages cannot be handled with SCIL.

8.5.3 The Custom Style Options

- **Tabs:** Specifies if the notebook pages should have tabs.
- **Tab Side**: Specifies the side of the notebook where the tab set should appear.
- **Folio**: Specifies whether or not to show the folio. The folio shows the page number at the bottom of the notebook pages.
- **Folio Justification**: Specifies the justification of the page number at the bottom of the notebook pages.
- **Page Buttons**: Specifies whether or not to show the page buttons.
- **Pages Wrap**: Specifies should the first page of the notebook be selected after the last one while moving to the right.
- **Tab Style**: Specifies the default border style of a tab.
- **Vertical Tabs**: Specifies whether or not to show vertical tabs.
- **Tab Buttons**: Centered Determines if the buttons to scroll the tabs are to be centered in the tab set rectangle or aligned with the inside border.
- **Tab Buttons**: Placement Determines where the buttons to scroll the tabs are displayed.
- **Allow Partial Tabs**: Specifies whether or not to show the tabs that are partially visible in a tab set.

8.6 Containers

Containers are mainly used to group dialog items that are somehow related to each other. For example, a set of option buttons can be placed into a container. See Figure 47. The **Container Attributes** page is discussed in this section. The common definitions were discussed in Section 7.

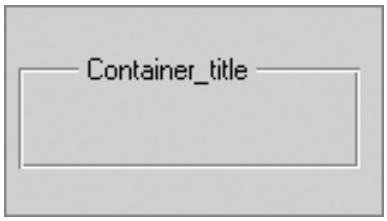


Figure 47: An example of a container

The containers can contain all types of dialog items, even other containers. The dialog items are added and arranged in the **Container** page. To add, edit and arrange the dialog items, follow the procedures described for dialogs in Section 7.

8.6.1 Container Attributes Page

The **Container Attributes** page shown in Figure 48 defines scroll bars for the container and keyboard traversal behavior. To insert scroll bars to a container, select the **Horizontal Scroll Bar** or **Vertical Scroll Bar** options. If the **Flat Keyboard Traversal** option is selected, the user can use the TAB key on the keyboard to move between items in the container and the other dialog items.

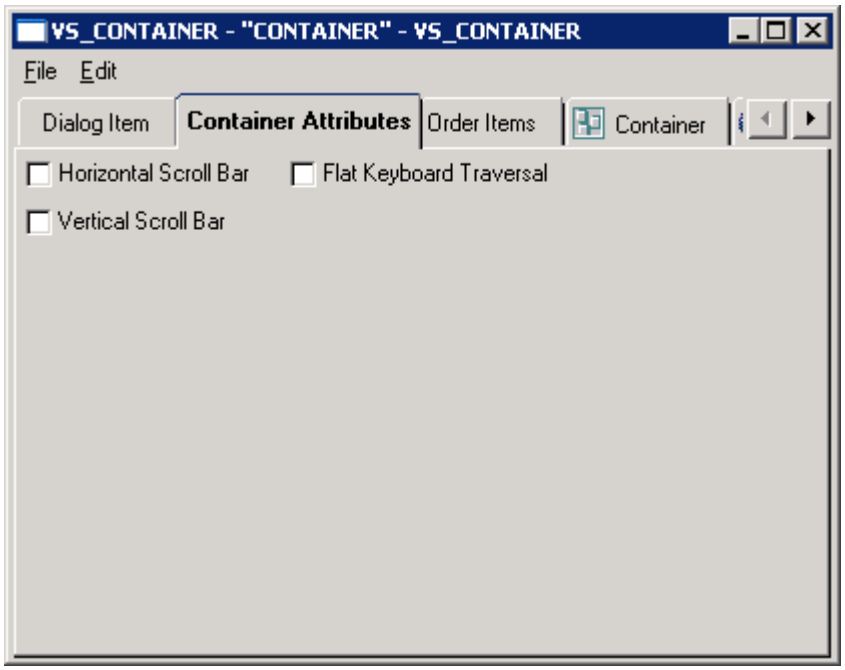


Figure 48: The Container Attributes page of the Container Object Editor

8.7 Menus

8.7.1 General

Menus provide the user a way to issue commands. There are three object types for menus:

- Menubar
- Menus
- Menu items

A menubar usually contains several menus, which all contain several menu items, see Figure 49. Begin to add menus by adding the menubar first. Then, add the menus and menu items.

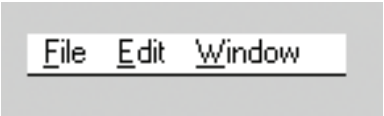


Figure 49: An example of a menu

8.7.2 Adding Menus

To define a menu:

1. Add the menu bar and enter its object editor.
2. Click **Menu Bar** to enter the page where the menus can be added. The page shown in Figure 50 appears.

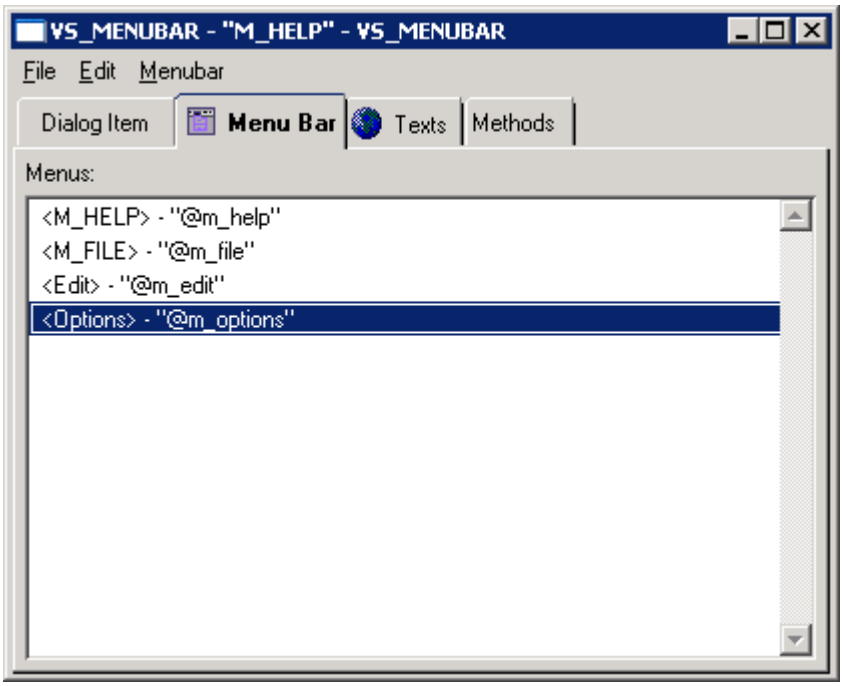


Figure 50: The Menu Bar page where you add new menus

3. Choose **New Menu** from the Menubar menu. A new menu appears. The name within brackets is the object name of the menu and the name within quotation marks is the title of the menu. The order of the menus can be changed by selecting the row and holding both mouse buttons down while dragging the row to a new position. The Has Help option on the Menubar menu means that the rightmost menu is placed at the right edge of the dialog on some systems. Otherwise, it is placed according to amount and size of menu items starting from the left side of the bar.
4. Open the Menu Object Editor by double-clicking the name of the menu, see Figure 51.
5. In the **Menu** page, define the title, name and mnemonic for the menu. The name of the menu is shown with a mnemonic in the upper left corner of the page.

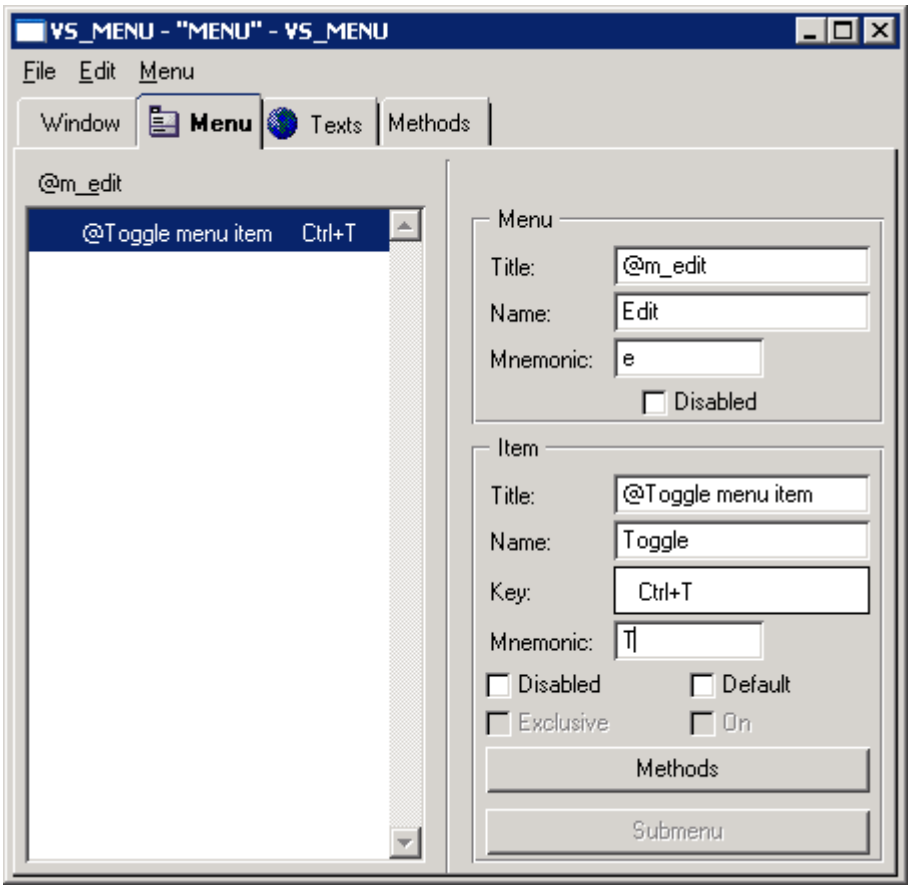


Figure 51: The Menu page in the Menu Object Editor where you add new menu items

6. To add a menu item, choose **New** from the Menu menu. On the submenu that appears, choose the menu item type you to be added. There are four types of menu items:
   - **Menu Item** is the regular command inserted in the menu.
   - **Separator Menu Item** is a line inserted between other menu items. It is the second menu item in Figure 51.
   - **Submenu Item** is the submenu item, which is a command displayed with an arrow pointing to the right. It indicates that more menu options are available when it is selected.
   - **Toggle Menu Item** is a menu item that is displayed with a check mark when it is selected.
7. Define the Title, Name, Key and Mnemonic for the new menu item. The name of the item is shown with the mnemonic key. The attributes Disabled, Default, Exclusive and On can also be defined depending on the type of the menu item that is being defined. The user can give more definitions in Object Editor of the menu item. To enter the editor, click **Programs**.

The On attribute can be selected for toggle menu items. If it is selected, the toggle menu item is selected by default and the check mark is shown in front of the option. The **Exclusive** option can also be selected for toggle menu items. If it is selected, only one toggle menu item in the menu, or in the part of the menu that is separated by a separator menu item, can be selected at a time.

If a submenu item has been added, the next step is to add menu items into the submenu. To add menu items into the submenu, enter the Submenu Object Editor and repeat steps 6 and 7.

8.7.3 Testing a Menu

In Menu Object Editor, the user can view and test a menu while defining it. To test a menu, choose **Try Out Menu** from the **Menu** menu. The features that have been defined are visible. The menu remains on the menubar until **Try Out Menu** is deselected.

8.8 Picture Containers

8.8.1 General

A Picture Container is a container used for conventional SYS600 pictures, pictures built in the Picture Editor. This way, pictures that are made in SYS600 Picture Editors can also be shown in Visual SCIL dialogs.

The upper left corner of the picture will be placed in the upper left corner of the picture container. If the picture container has scrollbars and the picture is bigger than the picture container, the user can move the picture around using the scrollbars. The used font and the size of the picture are determined by the current semigraphic font that was chosen when the application session was started.

The **Container Attributes** page shown in Figure 48 defines scrollbars and keyboard traversal behavior. To insert scroll bars into a container, select the **Horizontal Scroll Bar** or **Vertical Scroll Bar** options. It is not recommended to use the **Flat Keyboard Traversal** option in Picture Container.

8.8.2 Inserting a Picture

A picture can be inserted into the picture container using the picture handling methods. For example, methods can be in the Init method of a picture container, in a dialog, or in some button method. The place depends on the time the picture is to be shown. For more information on the subject, see Section 5

8.9 Icon View

The Icon View object is used for displaying icons. The icons are created with SCIL image editor. The predefined method _APPEND_ICON can be used for this purpose. For more information on adding icons, see Section 7. See also SYS600 Visual SCIL Objects and SYS600 Programming Language SCIL. Icon View Item is shown in Figure 52.

Figure 52: Examples of icon view object use, taken from the Tool Manager. The Dialog Editor icon is selected which apart from the highlighted text can be noticed as the Icon View area has focus.

Section 9 Other Dialog Items

9.1 About this Section

This section discusses all remaining dialog items, which are the items that cannot contain other dialog items. The definitions are done mainly in the object editor of the dialog item. If the user has added the dialog item inside another object, some of the effects of the definitions can be seen by looking at the Object Editor of the parent object. The test function described earlier in this manual can also be used. The procedure for adding objects is described in Section 6. The most common definitions, for example name and title, color and font setting, are described in Section 7.

9.2 Texts

9.2.1 Label

A label is a static text field, where the user cannot make changes. It is usually used to name another dialog item, see Figure 53. Defining the label does not contain any steps that are not common to other objects. The text to be shown to the user is written in the **Title** text box.

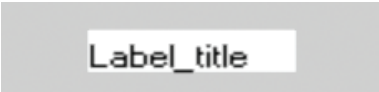


Figure 53: An example of a label

9.2.2 List

A list displays items organized into rows and columns, see Figure 54. Use the List Object Editor to define the appearance and behavior of the list. Usually, the content of the list is defined using SCIL, for example the _SET_CELL_TEXT method.

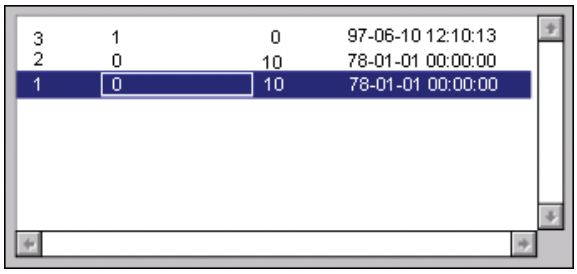


Figure 54: An example of a list

In the second page of the List Object Editor, the appearance of the list can be defined. This page is shown in Figure 55. In the Scroll Bars area, the user can select the **Horizontal** and **Vertical** options. **Vertical** is selected by default, so a vertical scroll bar is placed in the list. If **Horizontal** is selected, a horizontal scroll bar is placed in the list as well. In the **Show Grid** area, the user can choose Horizontal and Vertical grids, which draw lines between each row and column.

In the **Options** area, the options **Auto-Scroll**, **Traverse Wrap**, **Keyboard Selection** and **Keyboard Traversal** can be selected. **Auto-Scroll** is selected by default, so the user can move through a list by dragging the cursor to the edge of a list view to see the items beyond the

border of a list view. **Traverse Wrap** is not selected by default, so the user cannot move backward and forward from one end of the list to the other. **Keyboard Selection** is selected by default, so the user can select highlighted items in the list by pressing return. **Keyboard Traversal** is selected by default, so the user can move through the list by using the arrow keys on the keyboard.

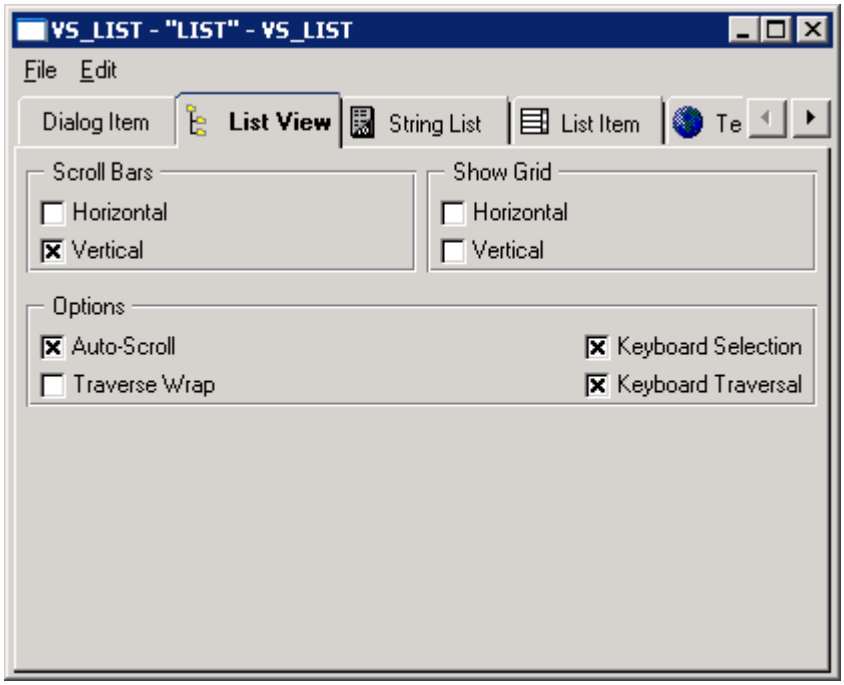


Figure 55: The List View page of the List Object Editor

The String List page of the List Object Editor is shown in Figure 56. It can be used to insert and remove rows and columns into a list. To insert a new row, click **Insert** under the text Row and to remove a row, click **Remove**. Inserting and removing columns is done in the same way under the text Column. The added rows and columns are shown in the white box with the column and row number in parentheses.

Usually, rows, columns, texts or other characters are set with SCIL. To insert texts or other characters into a cell manually:

1. Select the cell.
2. Click the **String** text box and type the text in it and press ENTER. The written text appears in the cell.

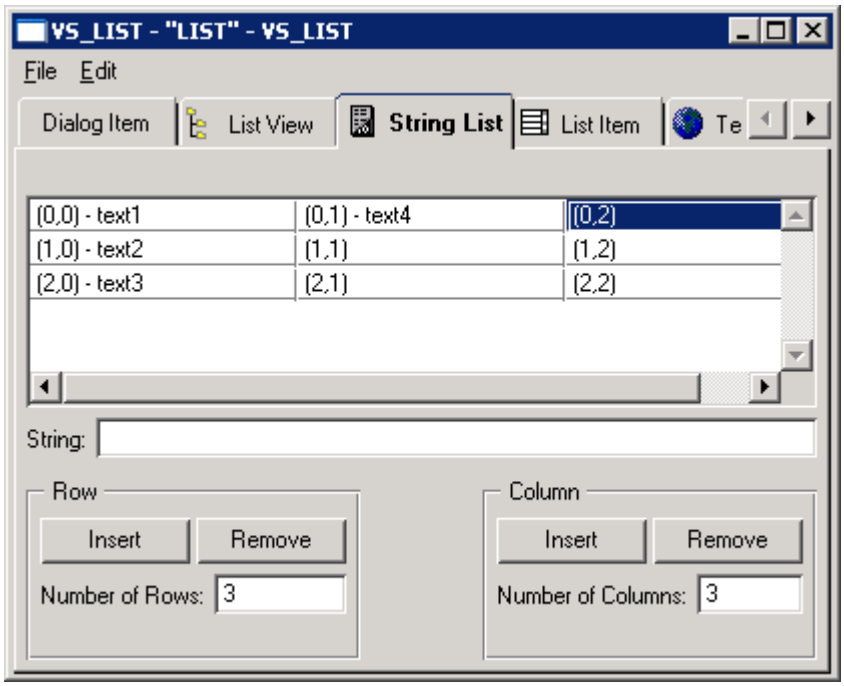


Figure 56: The String List page of the List Object Editor

The **List Item** page of the editor is shown in Figure 57. The **List Item** page can be used to specify the appearance and selection method for the list. Use the Natural Number of Rows and **Auto-Calculate Widths** options to determine the size of naturally sized list. A list can be defined as naturally sized by changing its inside connections. Natural Number of Rows is the number of rows used to calculate the natural height of a list. A naturally sized list displays the number of rows entered here. The value zero indicates that all rows are used to calculate the natural height.

The default column width is same as the width of the list. If **Auto-Calculate Widths** is selected, the default size is changed.

The selection method has four choices:

- **None** indicates that no cell from a list can be chosen.
- **Only One** indicates that no more than one cell can be selected at any time.
- **Free** indicates that any cell or a combination of cells can be selected.
- **Rectangular** indicates that the selection is always a rectangle of cells.

If the option **At Least One Selection** is selected, at least one cell in the list is always selected.

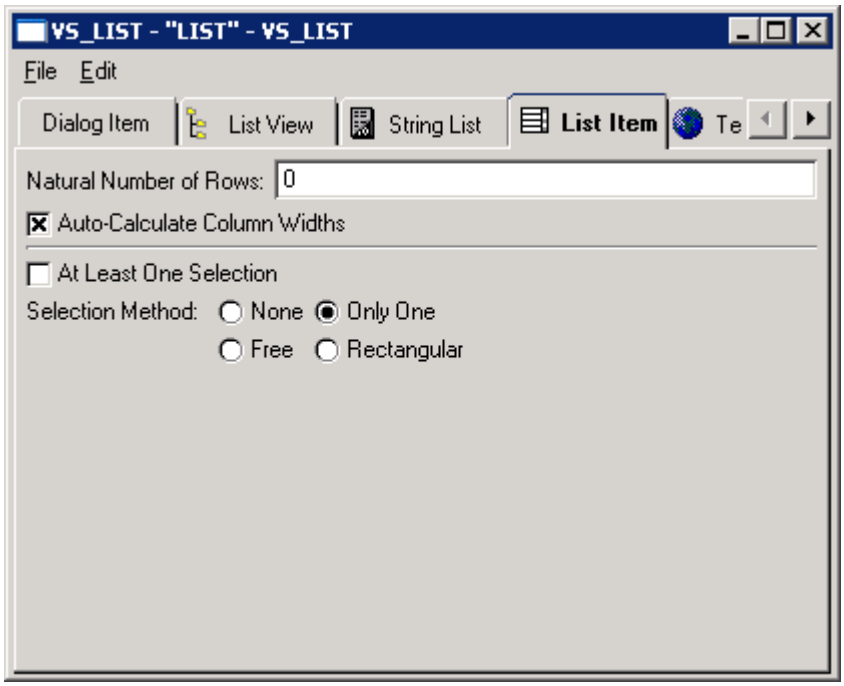


Figure 57: The List Item page of the List Object Editor

9.2.3 Text

In the text box, the user can insert or edit text. The operator can also define that the user cannot change the content of the text box, see Figure 58.

Figure 58: An example of text item

In the **Text View** page of the Text Object Editor, vertical and horizontal scroll bars can be added to the item. Select either one or both of the options in the **Scroll Bars** area. The Scroll Locked options makes it impossible for the user to move through the text horizontally or vertically. The user can also select a style for highlighted text and unhighlighted text. Highlighted text is selected text in an active window and unhighlighted text is the selected text in a window that is no longer active. The options are Default, Invert, Line, Box, None and Dotted line. The **Auto-Scroll** option is selected by default, and, usually, it should not be changed. When it is selected, the text in the text item is automatically scrolled.

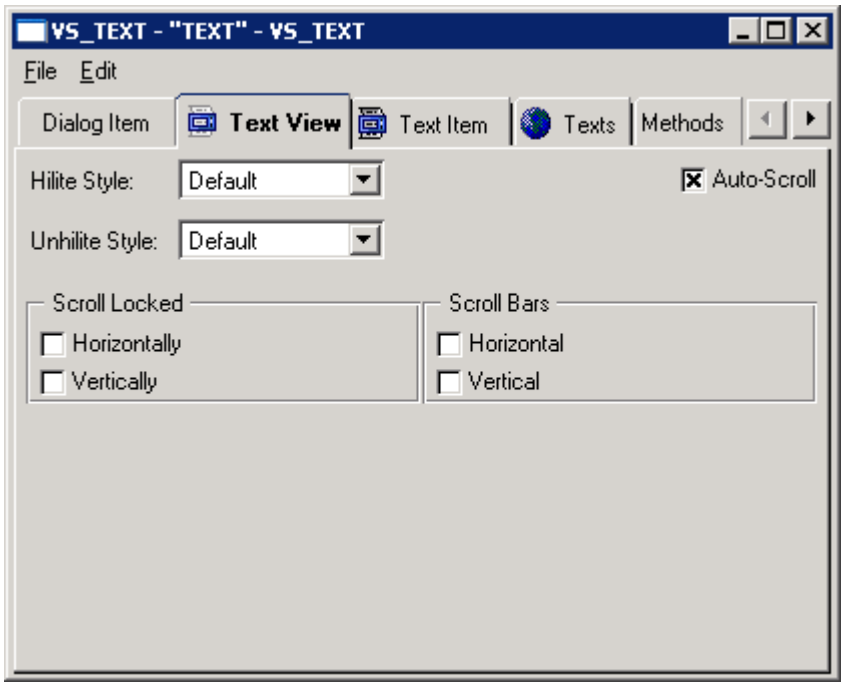


Figure 59: In this Text View page of the editor, you define how the text item behaves

The Text Object Editor contains a simple text editor with a text box and various features to specify the text properties, such as tab setting and line spacing. It is located in the **Text Item** page of the Text Object Editor, see Figure 60. The user can leave the large text box blank or place text in it. To type text, click in the text box and begin typing.

The **Selectable** option is selected by default, and it enables the user to select text in the text box. If this option is not selected the user cannot do this. The **Modifiable** option is selected by default. It enables the user to change the contents of the text box. If this option is not selected, the user cannot do this. The **Wrap** option is not selected by default. If it is selected, the editor automatically wraps the text. The **One Line** option is not selected by default. If it is selected, the text can be at maximum one line.

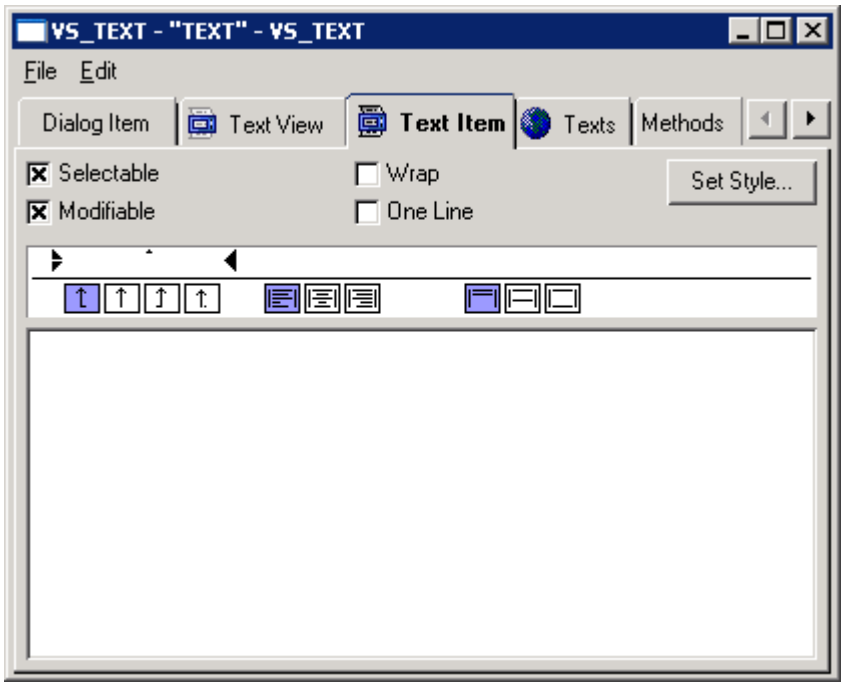


Figure 60: The Text Editor in the Text Object Editor

Use the text ruler, which is located under the check boxes, to set margins and tabs and to specify justification and line spacing for the text. To change the text style, click **Set Style**, and the Style Chooser shown in Figure 61 appears. Note that the changes made in the Style Chooser are only made to the selected parts of the text.

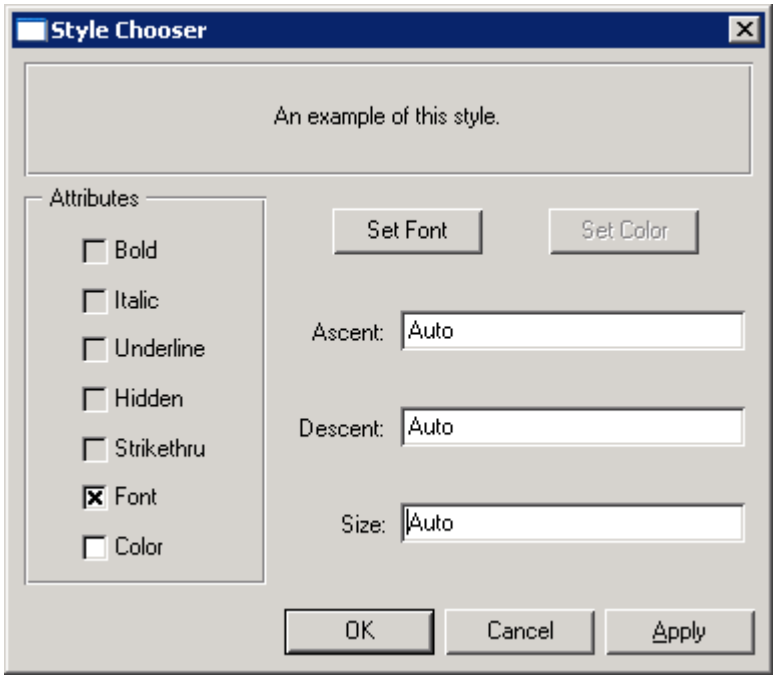


Figure 61: The Style Chooser where you can change the text style

The options that can be chosen with the check boxes are **Bold**, **Italic**, **Underline**, **Hidden** and **Strikethru**. These check boxes have three states: select, unselect and no selection. The unselect state removes the specified attribute. The no selection leaves the current attribute unchanged.

The Style Chooser also contains two buttons for setting Font and Color. Click either one and the Font Chooser or the Color Chooser appears. The procedure for using Font and Color Choosers is described in Section 7.

In the Style Chooser, the Ascent, Descent and Size of selected text can also be defined. Ascent is the amount the type rises above the base line. Descent is the amount type descends below the base line. Size represents the font size. These attributes are set to auto, which means that the default values of the font define the attribute. To change the default values, type the number of points in the box.

9.3 Buttons

9.3.1 Button

When the user clicks a button, an action is performed according to NOTIFY action method inserted in the button. If the button is naturally sized, it automatically adjusts its size to the Title text. See Figure 62.

Figure 62: An example of a button

9.3.2 Palette

A palette is a collection of buttons. Palette contains palette items, on which you can place icon to represent the available object or action. See Figure 63. You can add the icons with the Image Editor or SCIL. For more information on using the Image Editor, see Section 10.

Figure 63: An example of a palette

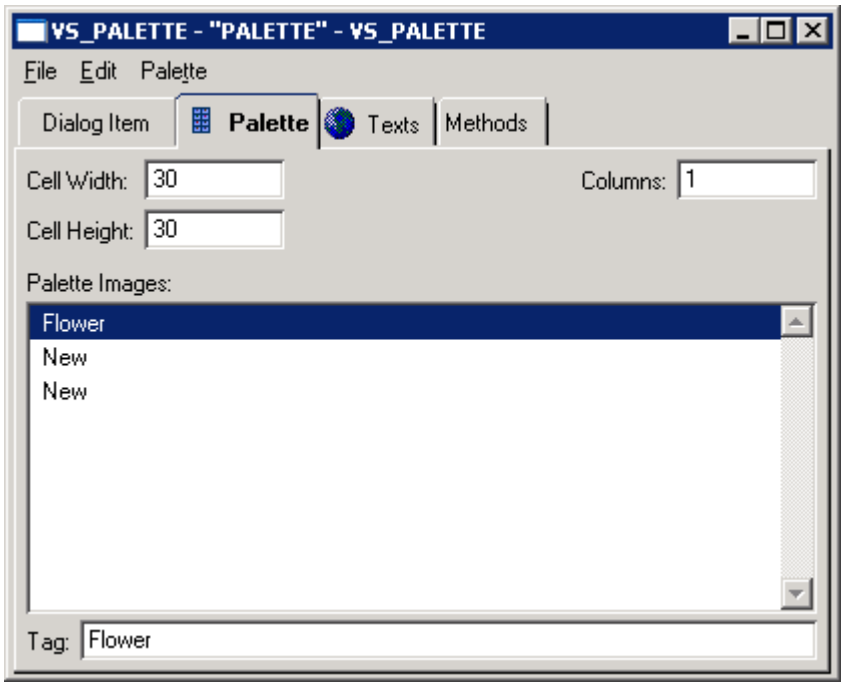


Figure 64: The Palette page where you can choose the palette items for the palette

To insert the palette items, choose **New Item** in the **Palette** menu. The item appears in the **Palette Images** text box. The name of the item is used only in this list, so it does not matter which name the item has. To open the Image Editor, double-click the item. The size of the palette item can be changed in the upper part of the editor. Click the **Cell Width** or **Cell Height** and type the new number of points. In the **Columns** box you can change the number of columns in the Palette.

9.4 Option Selection

9.4.1 Check Box

The user can select or unselect a check box of an option. To give the user several options from which all, none or some can be selected, the operator can use check boxes, see Figure 65. It may be useful to place check boxes in a container. A check box can have two values, On and Off. It has a set of states, which can be defined by control values. An example of a page in which control values for check boxes can be defined is shown in Figure 66.

Figure 65: An example of a check box

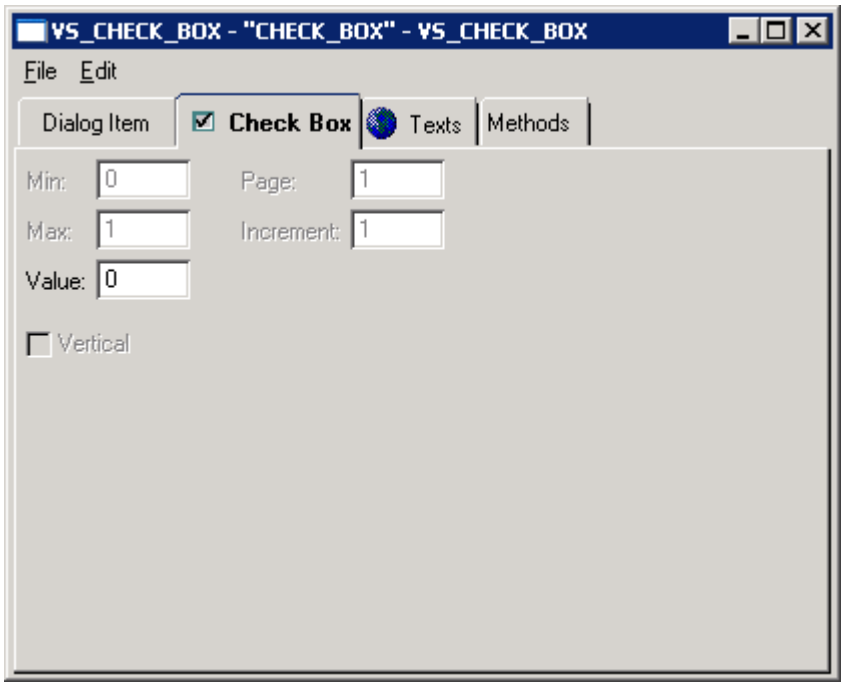


Figure 66: The control values for a check box are set in the Check Box page

The control values and their meaning while defining check boxes are listed here. The following attributes are described in more details in the SYS600 Visual SCIL Objects manual. See the attribute in the check box description.

- **Min** Does not change. The default value is 0. For more information, see the attribute _MIN_VALUE in SYS600 Visual SCIL Objects.
- **Max** Does not change. The default value is 1. For more information, see the attribute _MAX_VALUE in SYS600 Visual SCIL Objects.
- **Value** Defines the default value. The choices are On or Off. Value 0 means Off and value 1 means On. For more information, see the attribute _VALUE in SYS600 Visual SCIL Objects.
- **Page** Has no meaning for Check Box. For more information, see the attribute _PAGE_INCREMENT_VALUE in SYS600 Visual SCIL Objects.
- **Increment** Has no meaning for Check Box. For more information, see the attribute _INCREMENT_VALUE in SYS600 Visual SCIL Objects.
- **Vertical** Has no meaning for Check Box. For more information, see the attribute _VERTICAL_VALUE in SYS600 Visual SCIL Objects.

9.4.2 Option Button

Use option buttons to allow the user to select one of the given options. The title is placed on the right side of the button. See Figure 67.

The following attributes are described in more details in SYS600 Visual SCIL Objects. See the attribute in the description of the option button.

- **Min** Does not change. The default value is 0. For more information, see the attribute _MIN_VALUE in SYS600 Visual SCIL Objects.
- **Max** Does not change. The default value is 1. For more information, see the attribute _MAX_VALUE in SYS600 Visual SCIL Objects.
- **Value** Defines the default value. The choices are On or Off. Value 0 means Off and value 1 means On. For more information, see the attribute _VALUE in SYS600 Visual SCIL Objects.

Figure 67: An example of an option button

9.4.3 Combo and Combo Popdown

The word combo comes from combination box. A combination box is a list of texts the user can choose from. The difference between Combo and Combo Popdown is a drop-down list box. The Combo always displays the choices on screen and the Combo Popdown has a drop-down list box, which only shows the list when the user clicks the black arrow at the end of the box. The user can either type the selection in the box or select an option from the list. The possibility to type text depends on how the object behaviour is defined. See Figure 68 and Figure 69.

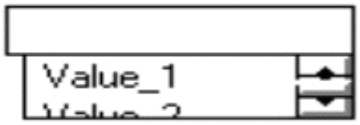


Figure 68: An example of a combo

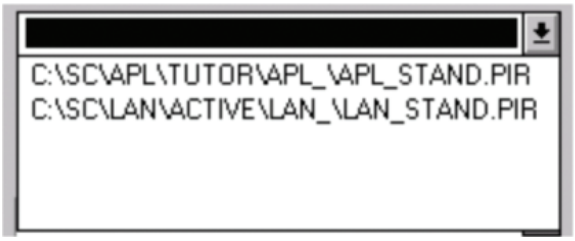


Figure 69: An example of a combo popdown

The appearance and the behavior of an object in the Combobox or Popdown can be defined in the **Combobox** page in their Object Editor. See Figure 70. Usually the contents are set with SCIL.

To add a row, choose **New** from the Combo menu. A new row appears.

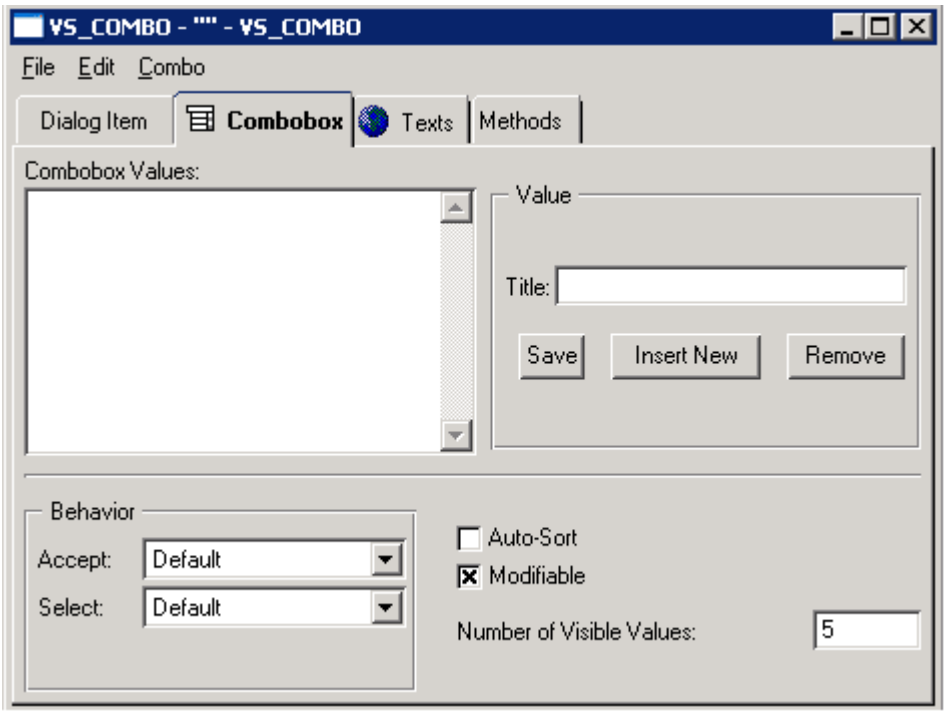


Figure 70: The Combobox page of the Combo Object Editor

To insert text or other characters:

1. Select the row.
2. Type the text in the Title text box and press ENTER. The input appears in the list.
   Save: Select the row in the list. Rename the text in the Title text box. Press **Save**. The renamed value appears in the list.
   Insert: Type the text in the Title text box. Press **Insert**. The input appears in the list.
   Remove: Select the row in the list. Press **Remove**. The selected row is deleted from the list.

The behavior of the combo is determined at the bottom of the page. There are eight choices to Accept, Select and Scroll situations. The choices are Do Nothing, Match or Add, Match or Closest, Match or Member, Match or Nothing, Match or First, Match or Revert and Default. Usually, the default setting can be used.

Normally, the text box is modifiable, which means that the user can enter a selection. If **Auto-Sort** is selected, the choices are automatically arranged in alphabetical order. The number of visible values defines how many options are shown on screen at any one time. The user can change the visible options by changing the number in the **Number of Visible Values** text box.

9.4.4 Numeric Spinner

A numeric spinner allows the user to enter values to define a range, see Figure 71. The user can move through possible values and view all of them, but cannot see them all at the same time. The user enters the value selecting the number from the list or by typing it in the box. Numeric spinners accept only integer values. The **Control** page for the Numeric Spinner is shown in Figure 72.

Figure 71: An example of a numeric spinner

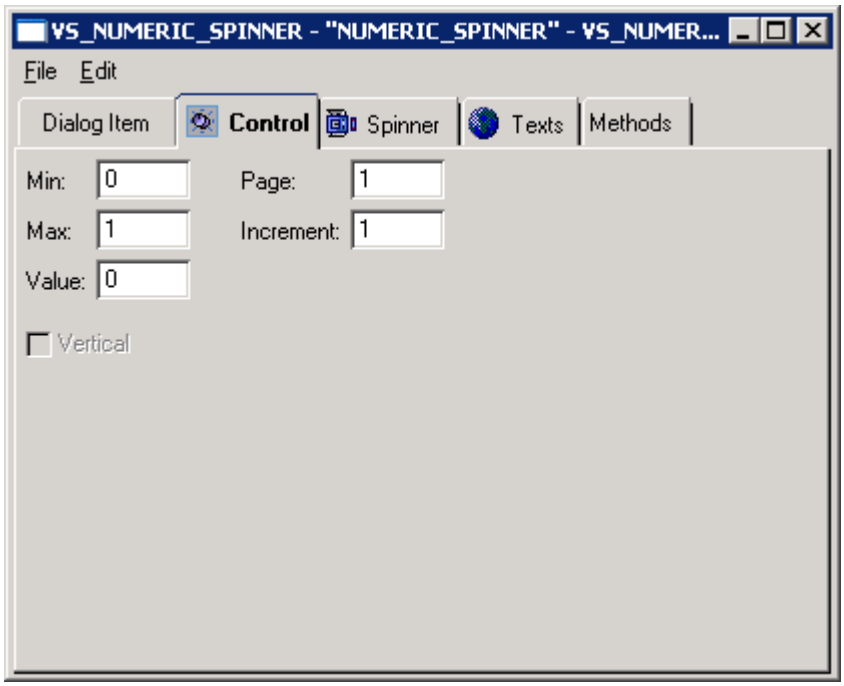


Figure 72: The Control page of the Numeric Spinner, where you can specify control values

The following attributes are described in more details in SYS600 Visual SCIL Objects. See the attribute in the description of the numeric spinner. In the **Control** page of the Numeric Spinner the user can specify:

- **Min** Minimum value defines the lowest value in the spinner range. For more information, see the attribute _MIN_VALUE in SYS600 Visual SCIL Objects.
- **Max** Maximum value defines the highest value in the spinner range. For more information, see the attribute _MAX_VALUE in SYS600 Visual SCIL Objects.
- **Value** Defines the default value of the spinner. For more information, see the attribute _VALUE in SYS600 Visual SCIL Objects.
- **Increment** The increment for a spinner indicates how much the value is increased or decreased when the arrow button is clicked. For more information, see the attribute _INCREMENT_VALUE in SYS600 Visual SCIL Objects.

Normally the text box of a numeric spinner is modifiable, which means that the user can enter a selection. If the **Wrap** option is selected, the user can scroll backward and forward from one end of the list to the other. Otherwise, the user can only scroll to the end of the list.

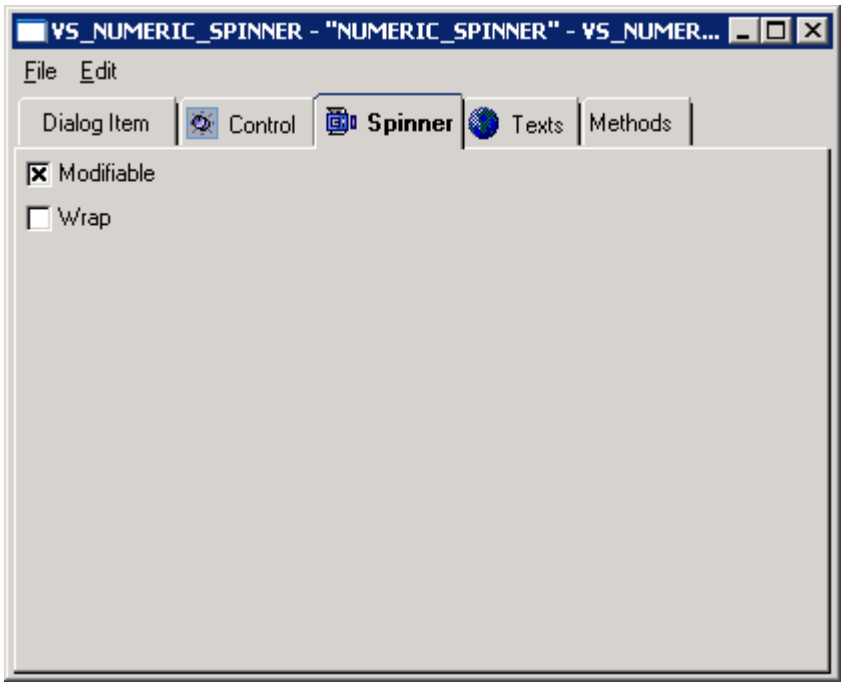


Figure 73: The Spinner page of the Numeric Spinner

9.4.5 Text Spinner

A text spinner allows the user to enter text from a defined range. The user can move through possible strings and view all of them, but cannot see them all at the same time. The user can enter the string by selecting it from the list or in some cases typing it in the **Text** box, depending on the definitions. If the user types a part of the string and presses enter, the first string that begins with those characters is shown. See Figure 75.

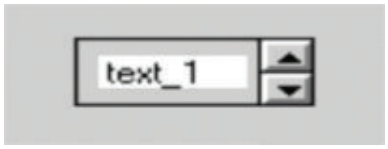


Figure 74: An example of a text spinner

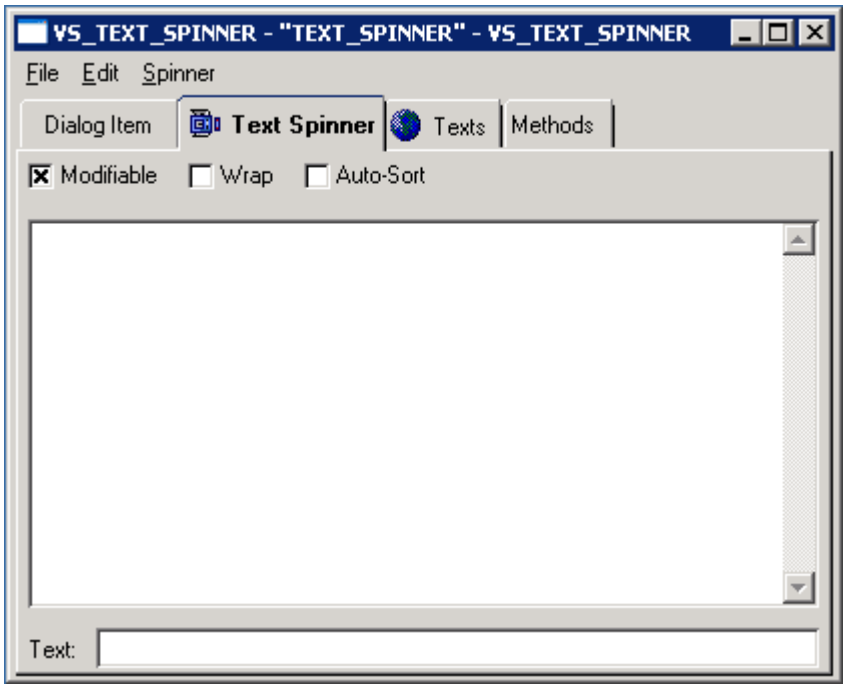


Figure 75: The options that the user can select in the text spinner

In the **Text Spinner** page of the Text Spinner Object Editor, the options to be shown in the spinner can be inserted. This can also be done with SCIL. To insert a new option, choose **New Entry** from the **Spinner** menu. A new string appears in the large white box.

To insert text or other characters in the string:

1. Click the string to select it.
2. Click in the **Text** box and type the text and press ENTER. The written text appears in the option.

To remove the option, click it and then choose **Clear** from the Edit menu. If the cursor is in the Text box, only the text in it is removed. Select the option in the large white box and try again to remove the option.

Normally, the text box of a text spinner is modifiable, which means that the user can enter the selection in the box. If the **Wrap** option is selected, the user can scroll backward and forward from one end of the list to the other. Otherwise, the user can only scroll to the ends of the list. To automatically arrange the options in alphabetical order, select the **Auto-Sort** option.

9.5 Scroll Bars and Sliders

9.5.1 Scroll Bar

By using a scroll bar, the user can move through a list or text. See Figure 76.

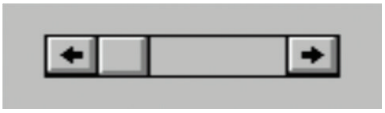


Figure 76: An example of a scroll bar

The following attributes are described in more details in SYS600 Visual SCIL Objects. See the attribute in the description of the scroll bar.

- **Min**: The default value is 0, which is the value of the starting point of the scroll bar. For more information, see the attribute _MIN_VALUE in SYS600 Visual SCIL Objects.
- **Max**: This is the maximum value or value at the end point of the scroll bar. The difference between the maximum and minimum values is the number of steps in the scroll bar. For more information, see the attribute _MAX_VALUE in Visual SCIL Objects.
- **Value**: Defines the default position of the scroll box. For more information, see the attribute _VALUE in SYS600 Visual SCIL Objects.
- **Page**: The number of steps the scroll box moves when the scroll bar is clicked. For more information, see the attribute _PAGE_INCREMENT_VALUE in SYS600 Visual SCIL Objects.
- **Increment**: The number of steps the scroll box moves when the scroll arrow is clicked. For more information, see the attribute _INCREMENT_VALUE in SYS600 Visual SCIL Objects.
- **Vertical**: If this option is selected, the scroll bar is vertical, otherwise it is horizontal. For more information, see the attribute _VERTICAL in SYS600 Visual SCIL Objects.

9.5.2 Slider

By using a slider, the user can select from a continuous range of possible values. The user drags the slider left or down to decrease the value and right or up to increase it. See Figure 77.

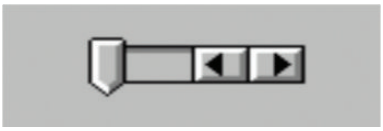


Figure 77: An example of a slider

The following attributes are described in more details in SYS600 Visual SCIL Objects. See the attribute in the description of the object you are defining. If the attribute is not described in it, it does not have effect on the object.

- **Min**: The default value is 0, which is the minimum value of the slider. For more information, see the attribute _MIN_VALUE in SYS600 Visual SCIL Objects.
- **Max**: This value is the maximum value of the slider. The difference between the maximum and minimum values is the number of steps in the slider. For more information, see the attribute _MAX_VALUE in SYS600 Visual SCIL Objects.
- **Value**: Defines the default position of the slider. For more information, see the attribute _VALUE in SYS600 Visual SCIL Objects.
- **Page**: The base unit by which the control unit is increased, so this is the number of steps the slider moves when some value of the slider is clicked. For more information, see the attribute _PAGE_INCREMENT_VALUE in SYS600 Visual SCIL Objects.
- **Increment**:The base unit by which the control unit is increased, so this is the number of steps the slider moves when it is dragged with the mouse. For more information, see the attribute _INCREMENT_VALUE in SYS600 Visual SCIL Objects.
- **Vertical**: If the option is selected, the slider is vertical, otherwise it is horizontal. For more information, see the attribute _VERTICAL in SYS600 Visual SCIL Objects.

9.6 Separating Items

9.6.1 Box

Use a box to group functionally related items or to improve the appearance of the dialog. A box is defined in the same way as other dialog items. A box can also be used to accommodate an image. See Figure 78.

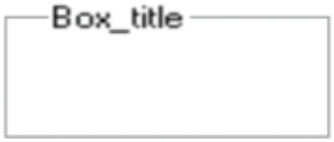


Figure 78: An example of a box

9.6.2 Icon View

Use the Icon View to display icons. The icons are created with SCIL. The predefined method _APPEND_ICON can be used for this purpose. For more information on adding icons, see Section 7. See also SYS600 Visual SCIL Objects and SYS600 Programming Language SCIL. Icon View Item is shown in Section 9.6.2.

Figure 79: In example of an icon view

9.6.3 Image Domain

An Image Domain is a rectangle in which images can be stored. It is recommended that it should be used only in advanced programming.

9.6.4 Line

Use the line dialog item to place a horizontal or vertical line into a dialog. It is often used to separate grouped dialog items. The line item is an item with a line drawn on one side. See Figure 80. The size of the line is changed by changing the size of the whole item, by moving the place of one of its border. This is done in the Object Editor of the object in which the line item is placed. The size of the line is drawn vertically by default. To have a horizontal line, change the size of the item so that the width of the line exceeds its height.

Figure 80: An example of a line

9.7 Tree

Use the Tree to display hierarchical data, see Figure 81. From the tree, it is possible to select and open different new objects, which are called nodes and leaves. The box can be added where the tree is drawn in the Tree Object Editor, but the tree itself is added with SCIL. The predefined methods _ADD_ROOT_NODE, _ADD_NODE and _ADD_LEAF are used for this purpose. Some action methods are also needed to define the behavior of the tree. The most often used action method is _HANDLE_EXPAND_NODE. For more information about adding trees, see Section 7, SYS600 Visual SCIL Objects and SYS600 Programming Language SCIL.

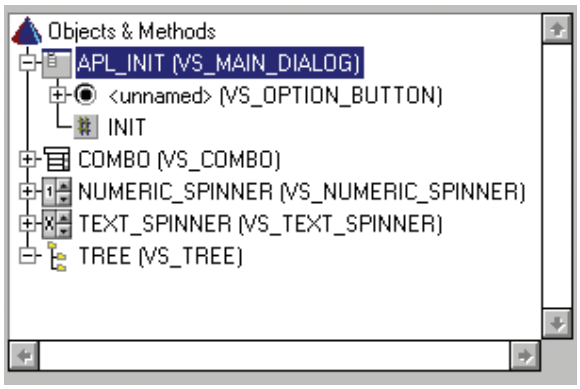


Figure 81: An example of a tree

In the scrollbars area the **Horizontal** and **Vertical** options can be selected. **Vertical** is selected by default, so a vertical scrollbar is placed in the tree. If **Horizontal** is selected, a horizontal scrollbar is placed. In the **Show Grid** area the user can choose Horizontal and Vertical grids, which draw lines between each row and column.

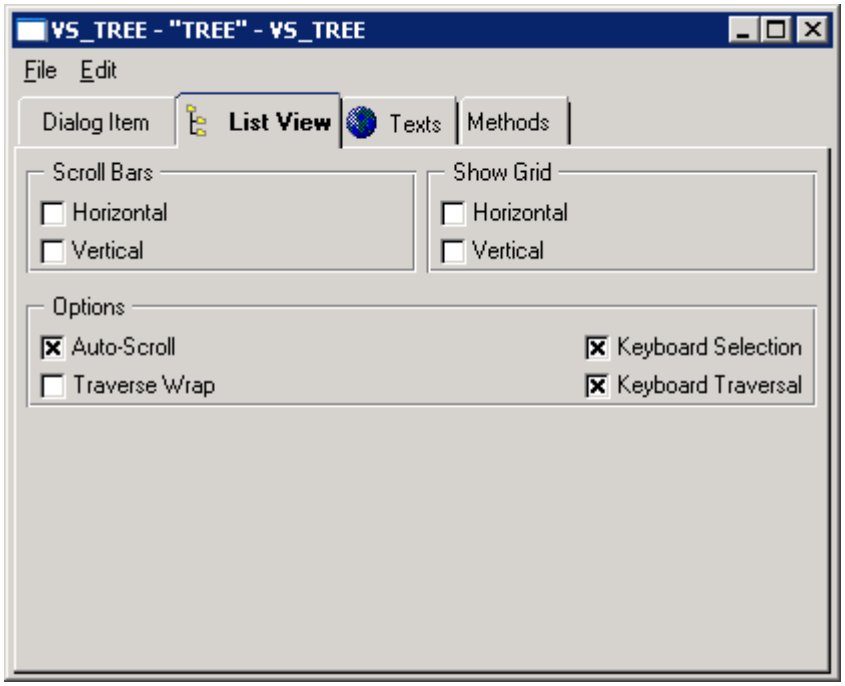


Figure 82: Define the special attributes related to the behavior of the tree

In the **Options** area, the following options can be selected:

- **Auto-Scroll**: Selected by default, so the user can move through a tree by dragging the pointer to the edge of a tree view to see the items beyond the border.
- **Traverse Wrap**: Not selected by default, so the user cannot move backward and forward from one end of the tree to the other.
- **Keyboard Selection**: Selected by default, so the user can select highlighted items in the tree by pressing Enter.
- **Keyboard Traversal**: Selected by default, so the user can move through the tree by using the arrow keys on the keyboard.

9.8 Header and Header Item

These Visual SCIL objects can be used as a header for any kind of column based data. The usage of these objects is based on both the VS_HEADER and the VS_HEADER_ITEM. A VS_HEADER is the main object that can contain any number of VS_HEADER_ITEMs. The user interacts with the header items and the programmer gets notification of the actions through the action methods of VS_HEADER.

The VS_HEADER object is added from the Dialog menu in the Dialog Editor. The items have to be created in a method. A suitable method to contain the creation statements of the items is the CREATE method of the VS_HEADER object.

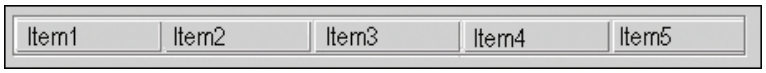


Figure 83: A VS_HEADER object with five VS_HEADER_ITEMs

The .create statements of the VS_HEADER_ITEMs placed in the CREATE method of the VS_HEADER object is shown below.

```
@l_geo = THIS._geometry
@i_width = l_geo:VW
@i_count = 0
.set THIS._arrange_enabled = false

#loop_with i= 1..5;(%i_number_of_columns)
.create item_'i' = vs_header_item
.set item_'i'._min_width = 0
.set item_'i'._max_width = 100
.set item_'i'._title = "Item'i'"
.set item_'i'._width = trunc(%i_width/5)
#loop_end

@b_created = true
```

Rearranging the items is managed by the attribute _ARRANGE_ENABLED and resizing the items is handled by the ITEM_RESIZED action method. A simple example of the ITEM_RESIZED action method is listed below.

```
@i_index = argument(1)
@i_width = argument(2)
.set item_'i_index'._width = %i_width
```

Whenever the whole header has been resized the HEADER_RESIZED action method is called. This action method can be used to adjust the items to the header.

For a complete list of predefined action methods and attributes of these objects, see SYS600 Visual SCIL Objects.

9.9 Graph and Graph Legend

The Visual SCIL object Graph is capable of visually displaying numerical data in different types of diagrams.

The available diagram types are:

- Plotted graph (line), see Figure 84.
- Area graph, see Figure 85.
- Bar graph, see Figure 86.
- Stacked bar graph, see Figure 87.
- Pie graph, see Figure 88.

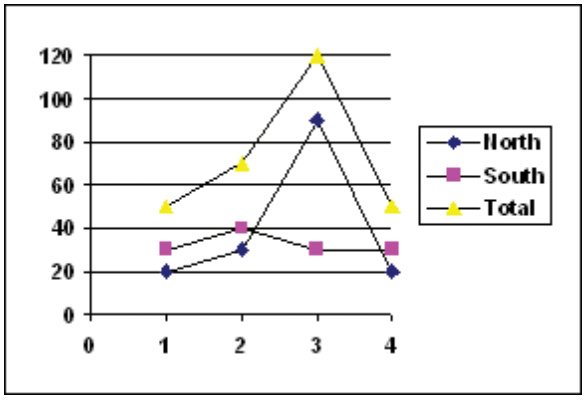


Figure 84: An example of a plotted graph (line)

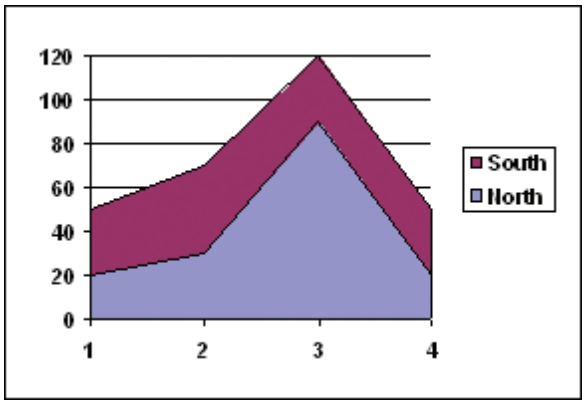


Figure 85: An example of an area graph

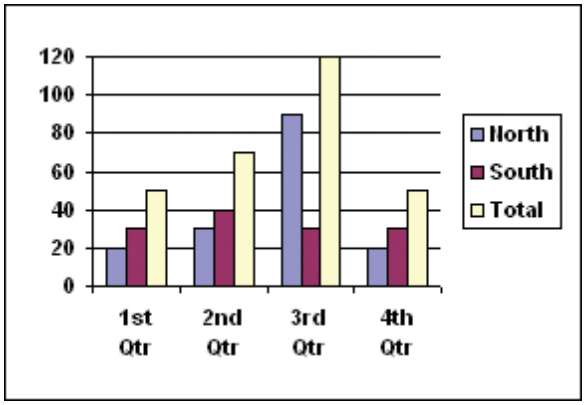


Figure 86: An example of a bar graph

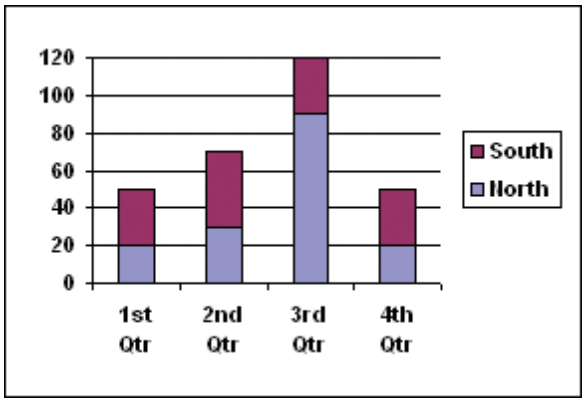


Figure 87: An example of a stacked bar graph

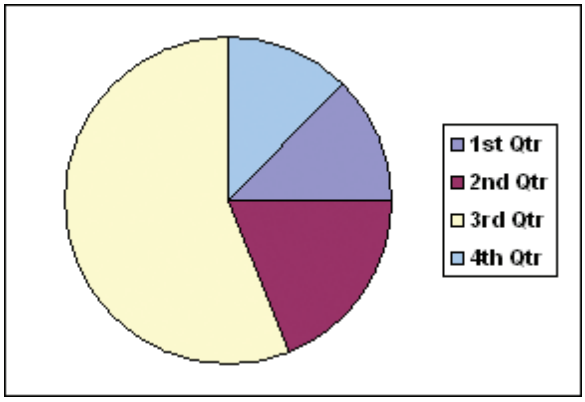


Figure 88: An example of a pie graph

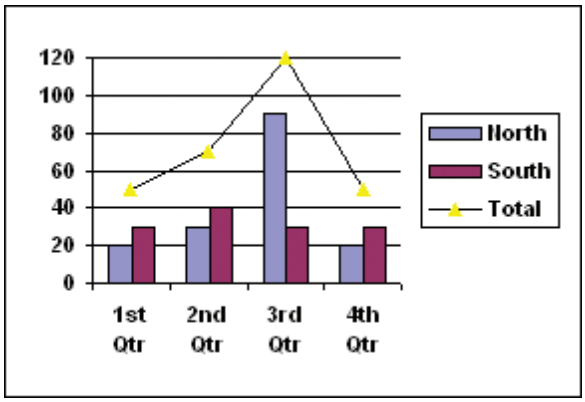


Figure 89: An example of a graph combining plotted and bar graphs

9.9.1 Datasets and Points

The data that is displayed in the graph is called a Dataset. A graph can display several Datasets. The individual values in the Dataset are called points.

9.9.2 Axis Annotation

The X-axis is annotated differently depending on the type of the X-axis. There are two types of X-axes used in graphing:

- Continuous
- Discrete

A graph's X-axis is continuous when spacing variations between points along the axis are important and discrete when spacing between the points are not important. For example, bar and stacking bar graphs always have a discrete X-axis.

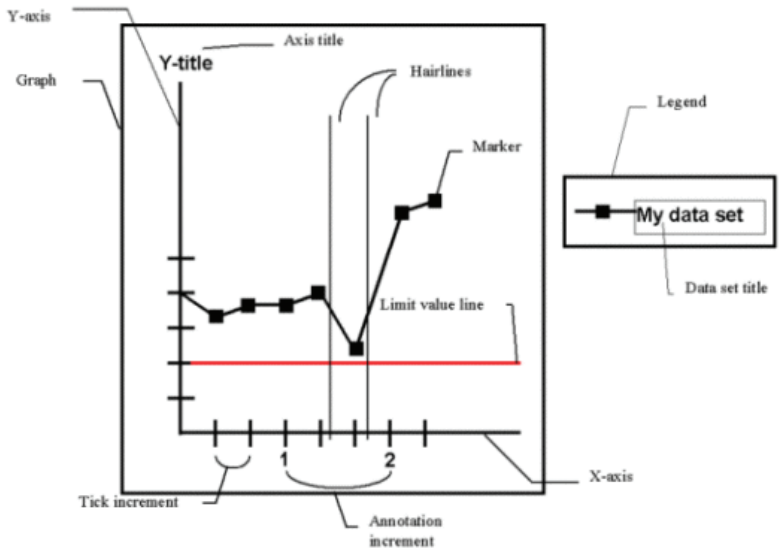


Figure 90: Terminology used in description of the VS_GRAPH object

9.9.3 Other Properties of the VS_GRAPH Object

Lines as markers for limit values can be displayed in the graph. An arbitrary amount of limit values can be assigned to either of the axes.

A rectangular area of the graph can be selected for zooming in on the data presented in the graph.

Hairlines can be moved horizontally over the graph. The intersection point between the graph and a hairline can be retrieved with a specific method of the VS_GRAPH object. If the orientation of the graph is inverted the hairline moves vertically.

For complete information on attributes and methods of the VS_GRAPH object, see SYS600 Visual SCIL Objects.

9.9.4 Using the VS_GRAPH Object

When using the VS_GRAPH object, certain methods have to be executed in a given order. The execution order of compulsory methods is:

1. Add X-axis
2. Set X-axis properties
3. Add Y-axis
4. Set Y-axis properties
5. Add dataset
6. Set dataset properties
7. Set dataset values

Each of these methods can be executed, for example, in the CREATE or INIT method of the VS_GRAPH object.

9.9.5 Example with VS_GRAPH Object

In this example data is collected from a data object. The used data object registers the current from a process object with 1 minute intervals and the RT attribute of the data object contains the time of each registration. This data is plotted as current versus time using the VS_GRAPH object.

The compulsory methods of the VS_GRAPH object are called from the INIT method of the object. A suitable range of values are read into a vector in a separate method, see Figure 91. Once the graph is drawn, updating is done in a cyclical method. The graph itself is updated as well as the properties of the X-axis. In case the Y-value would exceed the range on the Y-axis, its properties method could be added to the cyclically executed method.

Flow chart:

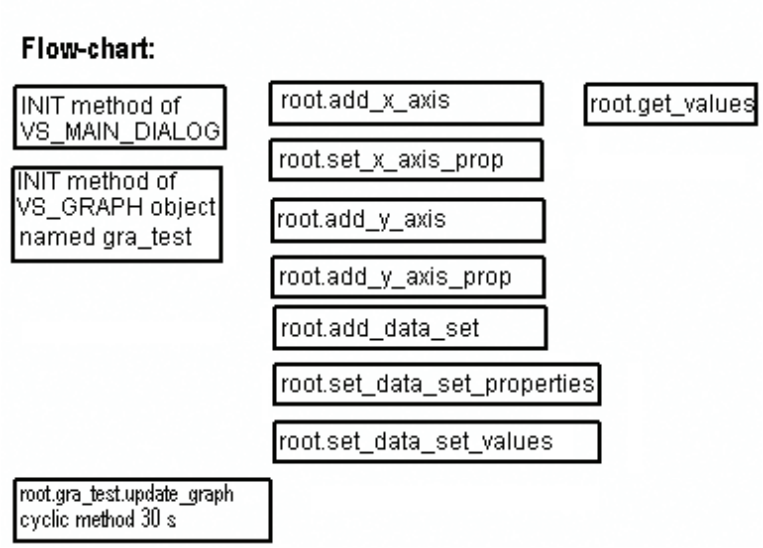


Figure 91: The execution order of essential methods in the example

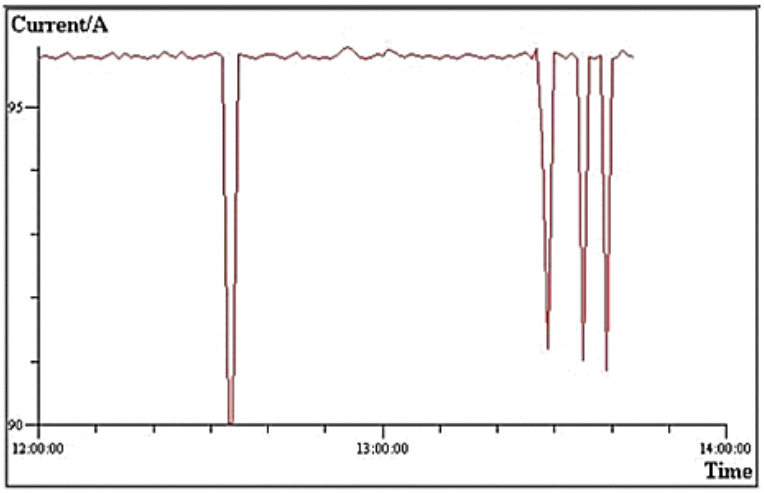


Figure 92: The graph of the example

Program listing of essential methods in the example:

```
;INIT method of VS_GRAPH object
root.add_x_axis
root.set_x_axis_prop
```

```
root.add_y_axis
root.set_y_axis_prop
root.add_data_set
root.set_data_set_properties
root.set_data_set_values

;add_x_axis method
gra_test._add_x_axis(1,"time","bottom",2,0)

;set_x_axis_prop method
@v_koord=.get_values
@v_xkoord=%v_koord(1)
@v_min_x=low(%v_koord(1))
@v_max_x=high(%v_koord(1))
@v_min_x(1)=%v_min_x(1)-((%v_min_x(1)) mod 3600)+2*3600
@v_max_x(1)=%v_max_x(1)-((%v_max_x(1)) mod 3600)+3600
@x_axis_prop_list=list( min=%v_min_x(1),-

max=%v_max_x(1),-
time_unit="SECONDS",-
time_format="%H:%M:%S",-
annotation_increment=3600,-
annotation_font=("M", 2),-
annotation_color="black",-
tick_increment=3600/6,-
axis_color="black",-
axis_title="Time",-
axis_title_font=("M", 1),-
axis_title_color="black",-
display_axis_title_enabled=TRUE,-
grid_mode="NONE",-
grid_line_width=1,-
grid_color="black",-
grid_line_style="SOLID")

gra_test._set_x_axis_properties(1, %x_axis_prop_list)

;add_y_axis method
gra_test._add_y_axis(2,"real", "bottom", 1,0)

;set_y_axis_prop
@v_koord=.get_values
@v_ykoord=%v_koord(2)
@v_min_y=(low(%v_ykoord))
@v_max_y=high(%v_ykoord)
@y_axis_prop_list=list( min=trunc(%v_min_y(1))/1.0,-

max=%v_max_y(1),-
annotation_increment=5,-
annotation_font=("M",2),-
annotation_color="black",-
tick_increment=1,-
axis_color="black",-
axis_title="Current/A",-
axis_title_font=("M",1),-
axis_title_color="black",-
display_axis_title_enabled=TRUE,-
grid_mode="NONE",-
grid_line_width=1,-
grid_color="black",-
grid_line_style="SOLID")

gra_test._set_y_axis_properties(2, %y_axis_prop_list)

;add_data_set method
gra_test._add_data_set(1,1,2)
```

```
;set_data_set_prop method
@data_set_prop_list=list(graph_type="PLOT",-

title="test",-
default_color="RED",-
show_line=TRUE,-
marker_shape="DOT",-
marker_color="BLACK",-
status_0_color="red",-
status_9_color="BLACK")

gra_test._set_data_set_properties(1,%data_set_prop_list)

;set_data_set_values method
@b_append=FALSE
@v_koord=vector()
@v_koord=.get_values
gra_test._set_data_set_values(1,%v_koord(1),%v_koord(2),%b_append)

;get_values method
@v_xkoord(%I)=vector()
@v_ykoord(%I)=vector()
@I=0
@i_last_index=(ftu_trd1:dlr)
#loop_with j= (%i_last_index - 180)..%i_last_index
@I=%I+1
@v_xkoord(%I)=(ftu_trd1:drt'j')-3.0*3600
@v_ykoord(%I)=ftu_trd1:dov'j'
#LOOP_END

#return vector(%v_xkoord, %v_ykoord)

;update_graph method, cyclical 30 s
root.set_x_axis_prop
;root.set_y_axis_prop
;Uncomment prev line if y-val out of range
root.set_data_set_values
```

9.10 Table

The VS_TABLE object is a two dimensional list of data that can have header information for each row and/or each column of that list. Instances of the VS_TABLE class can be used to display output information, like VS_LIST, but it can also be used to allow the user to enter new data or to modify existing data.

An example of a VS_TABLE is shown in Figure 93.

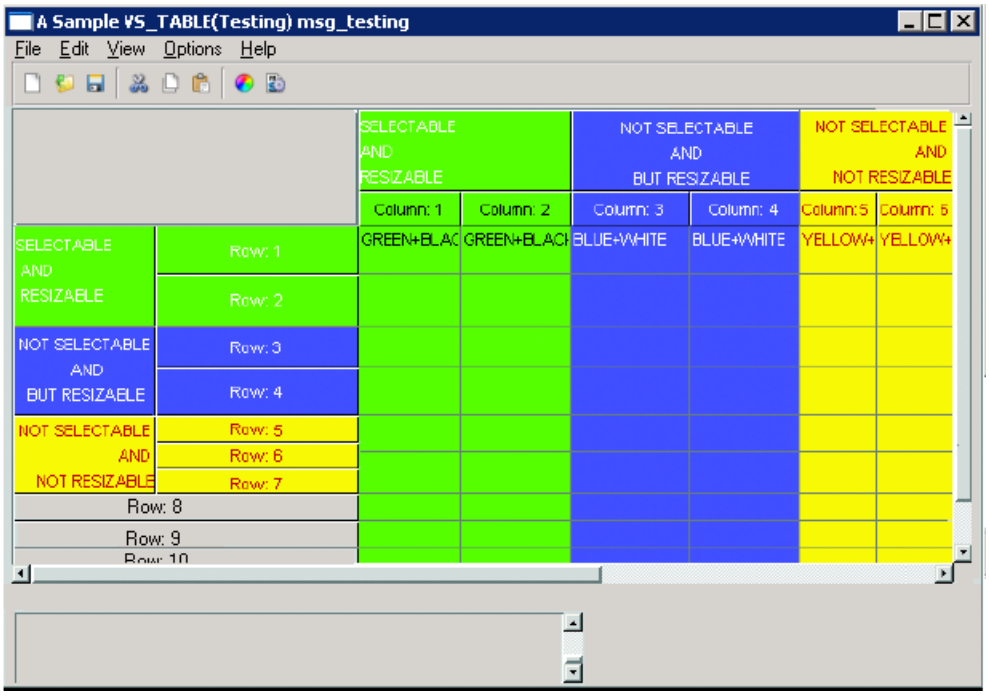


Figure 93: A simple VS_TABLE

The VS_TABLE object has about 50 attributes and 127 methods that allow developers maximal functionality. Many of these entry points were designed to ease the process of customization.

9.10.1 Defining the Appearance of the VS_TABLE

To define the appearance of the VS_TABLE:

1. In the **Text View** tab of the **VS_TABLE** dialog, select both **Horizontal** and **Vertical** in the **Scroll Bars** field.
   By default, the check boxes are checked.

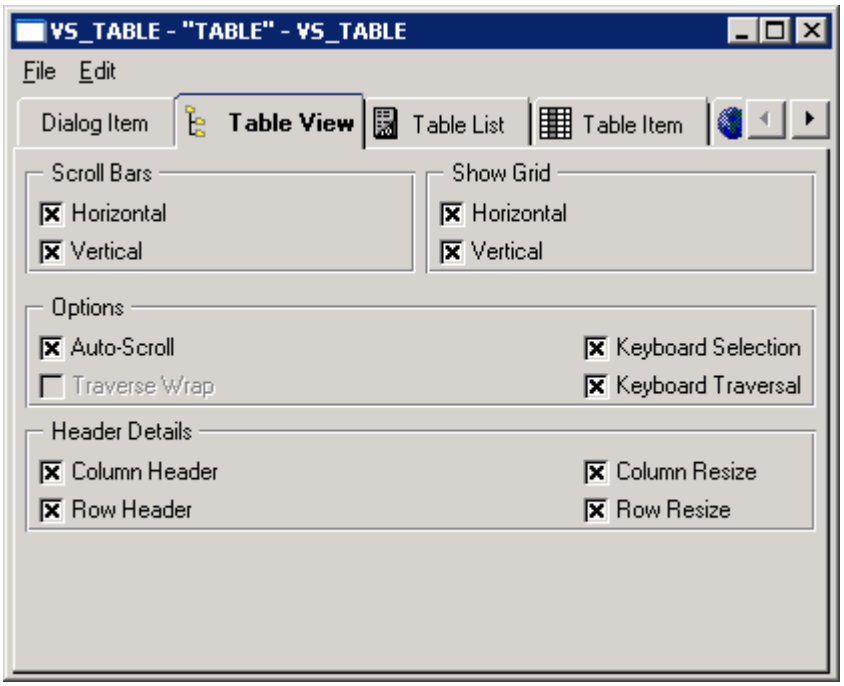


Figure 94: Defining the appearance of the VS_TABLE

2. In the **Show Grid** field, select **Horizontal** and **Vertical** to draw lines between each row and column.
3. In the **Options** field, select the options as necessary.
   - **Auto-Scroll** enables the user to move through a list by dragging the pointer to the edge of a table and see the items beyond the table border.
   - **Traverse Wrap** is disabled for a table. It can be enabled through SCIL.
   - **Keyboard Selection** enables the user to select highlighted items in the table by pressing ENTER.
   - **Keyboard Traversal** enables the user to move in the table using the arrow keys on the keyboard.
   - **Column Header** enables the user to have header to each column in the table.
   - **Row Header** enables the user to have header to each row in the table.
   - **Column Resize** enables the user to resize each column in the table.
   - **Row Resize** enables the user to resize each row in the table.

9.10.2 Inserting and Removing Rows and Columns

Rows and columns can be inserted and removed in the **Table List** tab of the **VS_TABLE Editor** dialog.

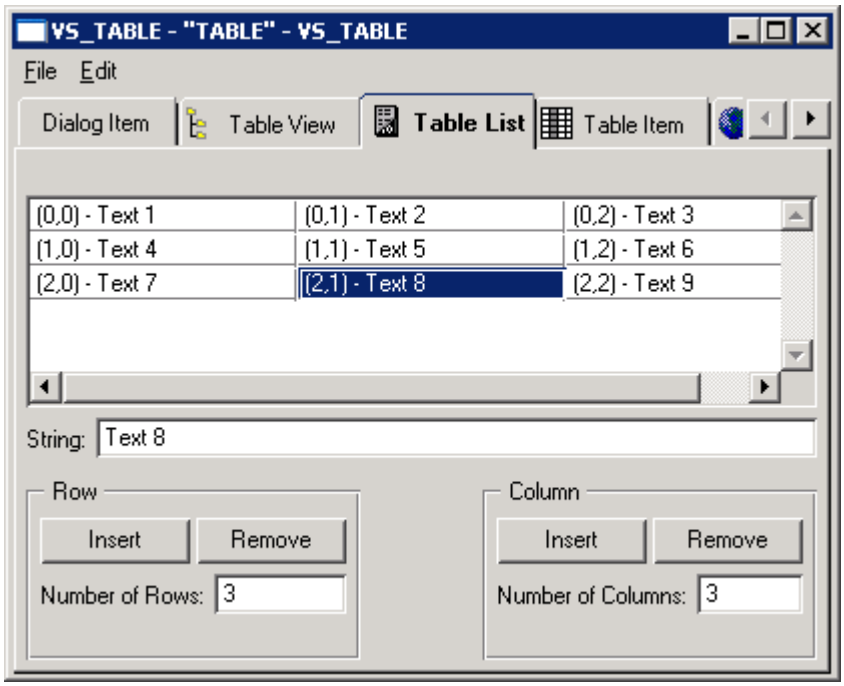


Figure 95: Inserting and removing rows and columns with the VS_TABLE Editor

To insert a new row, click **Insert**, and to remove a row, click **Remove** in the **Row** field. Insert and remove columns in the same way.

The added rows and columns are marked with a row and column number in parentheses.

The maximum number of rows is 10 000, and the maximum number of columns is 500.

9.10.3 Inserting Text into Table Cells

To insert text or other characters into a table cell manually:

1. Select the cell in the **Table List** tab of the **VS_TABLE Editor** dialog Figure 95.
2. Type the text in the **String** text input field, and press ENTER.

The new text can eb seen in the cell.

9.10.4 Specifying the Selection Method of Table Cells

To specify the selection method of table cells in the **Table Item** tab of the **VS_TABLE Editor** dialog, select one of the following:

- **None** indicates that no cell can be chosen from a list.
- **Only One** indicates that no more than one cell can be selected.
- **Free** indicates that any cell or a combination of cells can be selected.
- **Rectangular** indicates that the cells are always selected by forming a rectangle of cells with the mouse or keyboard keys.

At Least One Selection means that at least one cell in the list is always selected.

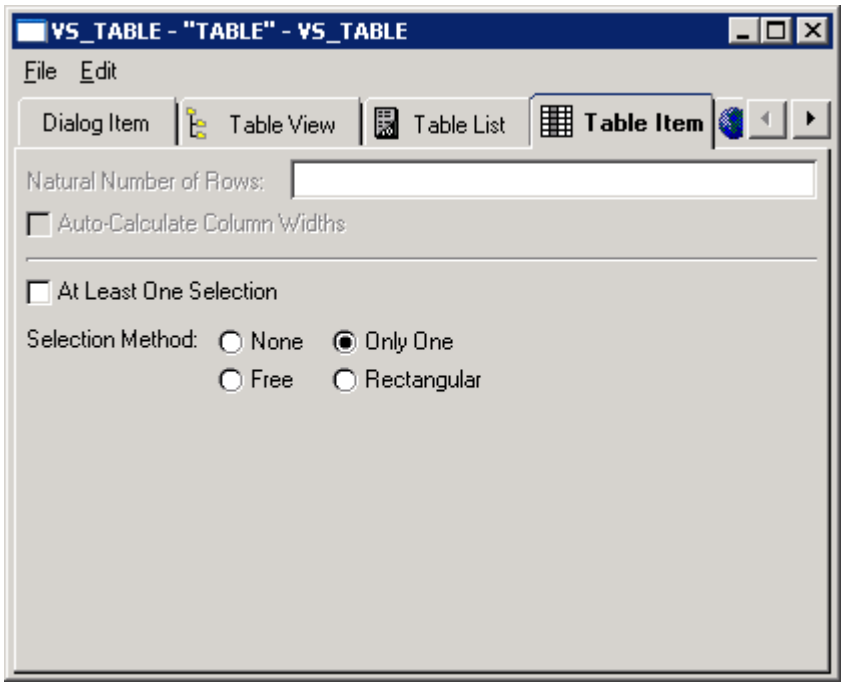


Figure 96: Specifying cell selection method with the VS_TABLE Editor

9.10.5 VS_TABLE Common Concepts

The VS_TABLE common concepts and their explanations are shown in Figure 97.

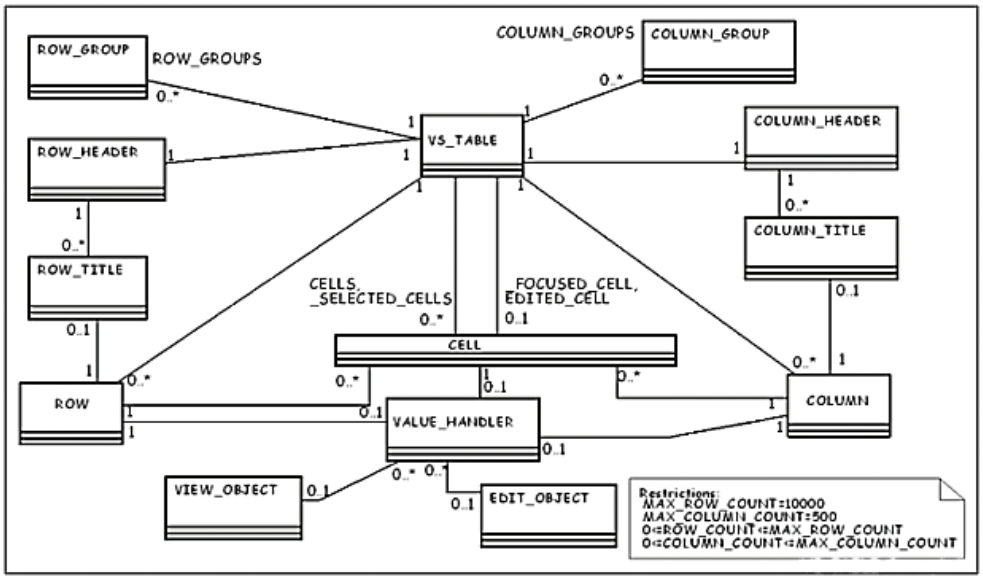


Figure 97: VS_TABLE common concepts and their relationships

Cell:

- Primary component, which is used to show user data.
- Has many programmable properties.

Row:

- Is used to define default properties for all cells in a given row.

Column:

- Is used to define properties for all cells in a given column.

Row and column title:

- Is used to define properties for a given row or column title.

Row and column header:

- Is used to define default properties for all row or column titles.
- Is used to define default properties for all row or column groups.

Row and column group:

- Is used to visually group rows and columns together.
- Is used to define properties for a row or column group.

Value handler:

- A mechanism to allow user to change the data of a cell.

Edit object:

- Allows the user to change the data of a cell.
- Different kind of Visual SCIL objects can be assigned to individual cells, entire row or column.

View object:

- Used for viewing the input data from the user.
- Different kind of Visual SCIL objects can be assigned to individual cells, entire row or column.

9.10.6 Initialising VS_TABLE

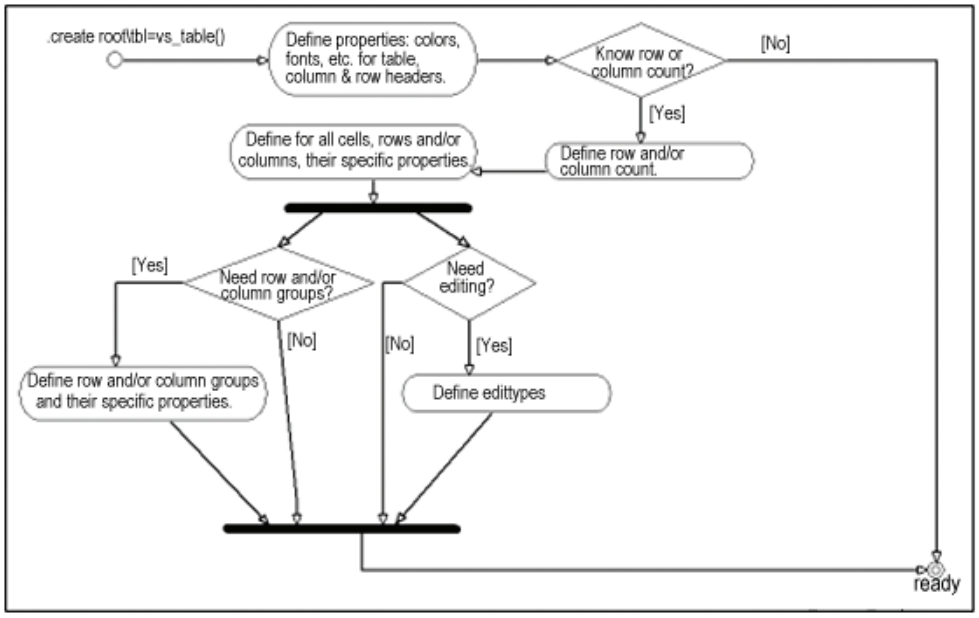


Figure 98: Initialising the VS_TABLE properties

The simplest case requires only that a VS_TABLE is created and some sensible attribute values are set. The following code shows an example of the creation of a simple VS_TABLE object with only the most common attributes set:

```
.create root\dlg=vs_dialog(_open=true)
.create root\dlg\table=vs_table()
.set root\dlg\table._column_count=100
.set root\dlg\table._row_count=100
root\dlg\table._set_column_title(1,"Column1")
root\dlg\table._set_column_width(1, 100)
```

A few points of interest demonstrated in the example code above:

- Line 2 creates a new instance of VS_TABLE.
- Lines 3 and 4 specify the number of columns and rows for this object.
- Line 5 and 6 specify the title for the first column as "Column1" with a width of 100.

9.10.7 Attribute Mechanism

Every cell, row, and column in the VS_TABLE object has a set of attributes associated with it that describes the content and appearance. Since it is often so that the same attribute value applies to all the cells in a given row or column, it is possible to specify, with a single call, an attribute value that will be applied to all the cells. This method of assigning attributes leads to a natural hierarchy of inheritance, which allows quite a lot of flexibility in programming. The diagram in Figure 99 illustrates the order of inheritance and provides a SCIL example of how to supply attributes at each level.

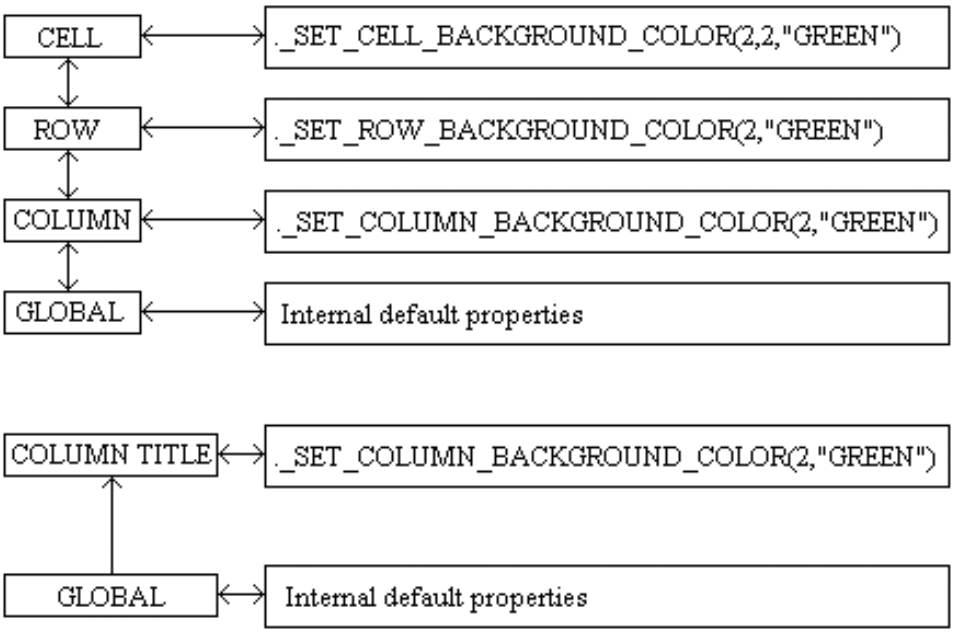


Figure 99: Attribute Inheritance and some examples

This attribute inheritance is utilized throughout the VS_TABLE object, including the row and column header attributes. VS_TABLE has methods to access the attribute values of specific cells, rows and columns. These methods look backwards up the inheritance path until a valid value is found.

9.10.7.1 Clearing the Table

The easiest way to clear the context of the VS_TABLE is to set column and row count to zero. In this case, the following attributes are initialized to their default values:

- When column count is set to 0:

```
default column title "Column: %d"
default column width 50
.set root\dlg\table._COLUMN_DRAG_AND_DROP_ENABLED=FALSE
.set root\dlg\table._COLUMN_HEADER_RESIZABLE=TRUE
.set root\dlg\table._COLUMN_HEADER_SELECTABLE=TRUE
.set root\dlg\table._EXTEND_COLUMN_SELECTION_ENABLED=FALSE
.set root\dlg\table._HAS_COLUMN_HEADER = TRUE
.set root\dlg\table._SORTING_ENABLED=FALSE
```

- When row count is set to 0:

```
default row title "Row : %d"
default row height = to a font dependent value
_EXTEND_ROW_SELECTION_ENABLED=FALSE
_HAS_ROW_HEADER=TRUE
_ROW_DRAG_AND_DROP_ENABLED=FALSE
_ROW_HEADER_RESIZABLE=TRUE
_ROW_HEADER_SELECTABLE=TRUE
_SORTING_ENABLED=FALSE
_ROW_HEADER_WIDTH=25
```

9.10.8 Cells

The cell is one of the main concepts of the VS_TABLE object. It encapsulates the various attributes associated with the cells of a VS_TABLE object.

Table 1: Basic cell attributes

Attribute	Type	Description
Background color	Color	The background color of the cell.
Font	Font	The font used when drawing the value of the cell.
Foreground color	Color	The foreground color of the cell.
Justification	Text	DEFAULT_JUSTIFY, CENTER_JUSTIFY, LEFT JUSTIFY, RIGHT_JUSTIFY.
Object id	Text	User defined free text.
Read-only	Boolean	TRUE if the cell cannot be edited.
Text	Text	The value of the cell.
Value handler		A value handler of the cell.

Several of the attributes in Table 1 affect the way the content of a cell is drawn. These are the background color, foreground color, font, and justification attributes.

The value or content of a cell is a string that is stored in the text attribute. If it is necessary to store additional information for a cell, there is an object id attribute that is text and can be used for this purpose. By default, all cells are considered read only and cell text is an empty string.

9.10.9 Titles, Headers and Groups

9.10.9.1 Row and Column Headers

VS_TABLE object supports row and column headers that give additional functionality to the overall object. The headers are used to define common properties and behavior of all row or column titles. They can be displayed or hidden from view. They can be marked as being selectable, which would allow the user to select a title and have the respective row or column selected. It is also possible to set the headers so that the user can interactively resize the rows and/or columns.

9.10.9.2 Row and Column Titles

It is usual that the default behavior of headers must be overridden for some column or row. Modifying a row or column title one by one does this.

9.10.9.3 Multiple Lines of Text in a Title

All titles have the special feature of being able to display multiple lines of text. Each line of the title is subject to the current justification and font settings for that specific title.

To separate a title into multiple lines simply add new line characters to the string assigned to the title. If a blank line is desired, at least one white-space character has to be added to the line due to a limitation in the parsing routine. The VS_TABLE object will divide all the available height of the header equally between all the lines in the title (see Figure 100).

An example of dividing the column title to multiple lines (see Figure 100):

```
._set_column_title(9,"Micro"+ASCII(10)+"SCADA"+ASCII(10)+"%d")
```

9.10.9.4 Automatic Numbering

The most common row title is the row number. There is a special case supported that allows the current row number (one based) to be included in the row or column title without having to specifically set a title for each row of the VS_TABLE object. In order to accomplish this automatic numbering, include the characters "%d" somewhere in the title and assign the same string to all the rows.

The default row title is "Row: %d" and the default column title is "Column: %d".

An example of automatic row numbering:

```
.create root\dlg=vs_dialog(_open=true)
.create root\dlg\table=vs_table()
.set root\dlg\table._column_count==5
.set root\dlg\table._row_count=10
root\dlg\table._set_column_width(%currentRow, 100)
root\dlg\table._set_column_title(%currentRow,"MicroSCADA")
#loop_with currentRow = 1 .. root\\tbl._row_count
root\dlg\table._set_row_title(%currentRow,"My row %d")
#loop_end
```

9.10.9.5 Column and Row Groups

There are many situations where it is preferable to group several columns or rows together under a single title. There can be many groups in many different levels.

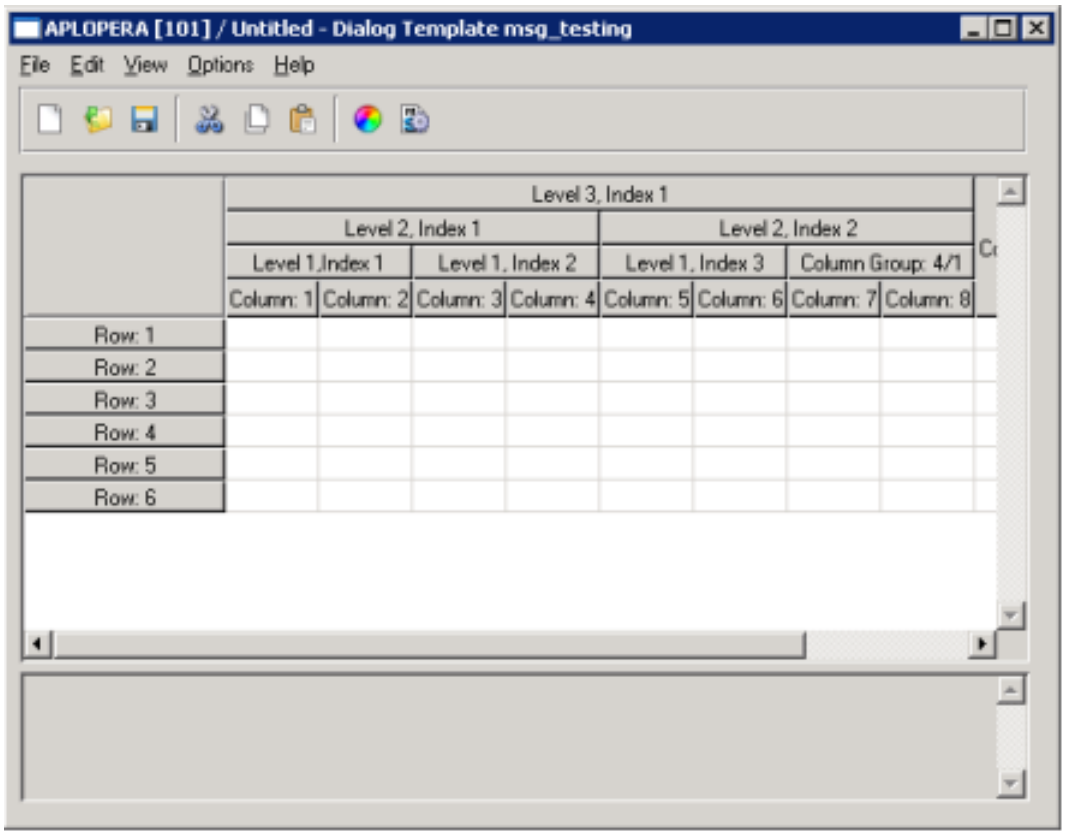


Figure 100: A VS_TABLE with some column groups defined

Because SCIL does not have pointers, there must be some mechanism to set group's properties. There is an indexing mechanism for accessing a given group. The table in Figure 100 has some column groups, which are created as follows:

```
root\dlg\table._add_column_group(1,2,1)
root\dlg\table._set_column_group_title(1,1,"Level 1,Index 1")
root\dlg\table._add_column_group(3,4,1)
root\dlg\table._set_column_group_title(1,2,"Level 1, Index 2")
root\dlg\table._add_column_group(5,6,1)
root\dlg\table._set_column_group_title(1,3,"Level 1, Index 3")
root\dlg\table._add_column_group(7,8,1)
root\dlg\table._add_column_group(1,4,2)
root\dlg\table._set_column_group_title(2,1,"Level 2, Index 1")
root\dlg\table._add_column_group(5,8,2)
root\dlg\table._set_column_group_title(2,2,"Level 2, Index 2")
root\dlg\table._add_column_group(1,8,3)
root\dlg\table._set_column_group_title(3,1,"Level 3, Index 1")
```

A few points of interest demonstrated by the above code:

- Line 1 creates a column group at level 1, which contains columns [1..2]. This particular group is at level 1, which means it will be displayed immediately above the normal column titles.
- Line 2 changes the column group title at level 1 index 1.
- Line 7 creates a column group, which will have a default column group title.
- Line 8 creates a column group at level 2, which contains columns [1..4].
- Line 12 creates a column group at level 3, which contains columns [1..8].

Due to the drawing and freezing reasons, there are some restrictions how a group can be defined. A new group is not allowed to include a part of lower level group. For example, the following lines will raise a SCIL error:

```
._add_column_group(1,3,1)
._add_column_group(2,7,2)
._add_column_group(1,7,4)
```

Here is an example of how to access column groups and to change all background and foreground colors at known level:

```
@level=1
#loop_with i = 1 .. root\dlg\table._get_column_group_count(%level)
root\dlg\table._set_column_group_foreground_color(%level,%i,"BLACK")
root\dlg\table._set_column_group_background_color(%level,%i,"RED")
#loop_end
```

Sorting is not possible when there are some groups defined.

9.10.9.6 Drag and Drop

In VS_TABLE it is possible to allow the user to change the order of rows and/or columns. This can be done using the following attributes:

- _COLUMN_DRAG_AND_DROP_ENABLED
- _ROW_DRAG_AND_DROP_ENABLED

By default, the drag and drop feature is disabled. When the drag and drop is enabled, for example, for columns, the user can do the drag and drop function with the mouse:

1. Move the mouse pointer over a column title or column group title.
2. Click down the left mouse button, the cursor changes.
3. Keep the mouse button down and move the column or column group to the new position.

If the column or row is inside a group, it can be dragged and dropped only inside of the group. If the column or row group is inside a higher level group, it can be dragged and dropped only inside of the higher level group

9.10.10 Introducing Editing Capabilities

One of the most powerful features of the VS_TABLE object is the ability to edit the values of the cells directly. In order to make a cell directly editable, it needs to be assigned a value handler and to have its read only attribute set to FALSE.

There is one predefined edit type included in the VS_TABLE object. There are also a few methods that can be used to define more complicated value handlers.

See Section 9.10.13 for more details on how to make cells editable.

9.10.11 Action Methods

There are several action methods, which can be programmed to react to user actions. See Section 9.10.14 to see the execution order of some of these methods. See also SYS600 Visual SCIL Objects for detailed descriptions.

9.10.11.1 CELL_EDITING_ACCEPTED

This action method is executed whenever the user has accepted a new value of an edited cell.

9.10.11.2 CELL_EDITING_CANCELLED

This action method is executed whenever the user cancels the editing of a cell.

9.10.11.3 CELL_EDITING_STARTED

This action method is executed whenever the user is starting the editing of a cell.

9.10.11.4 COLUMN_TITLE_CLICKED

This method is executed whenever the user clicks on the column title.

9.10.11.5 COLUMN_TITLE_DOUBLE_CLICKED

This method is executed whenever the user double clicks on the column title.

9.10.11.6 DRAGGED_AND_DROPPED

This method is executed whenever the user has dragged and dropped a column/row or multiple columns/rows.

9.10.11.7 FOCUSED_CELL_CHANGED

This action method is executed whenever the user has changed the focused cell with mouse or arrow keys.

9.10.11.8 SELECTION_HAS_CHANGED

This action method is executed whenever the selection is going to change, is changing and has changed.

9.10.11.9 TOOLTIP_IS_SHOWN

This action method is executed whenever a tooltip is shown over the dialog item.

9.10.12 Miscellaneous Features

9.10.12.1 Column Freezing

It is possible to freeze a range of columns. Frozen column is always at the visible area of the table.

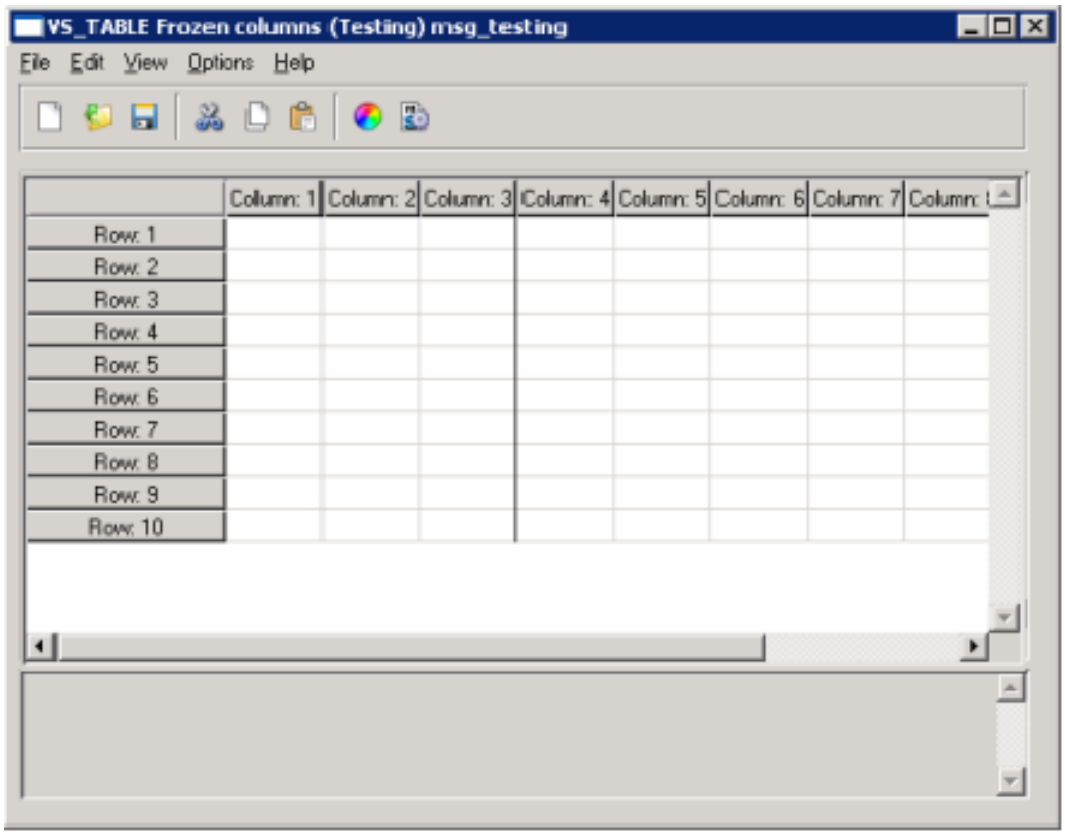


Figure 101: A VS_TABLE with three frozen columns

Example:

```
._freeze_columns(3)
```

A valid range of columns can be frozen [1..user specific]. If a column group is needed, only one whole column group in level 1 is accepted, otherwise a SCIL error is raised.

An example using a column group:

```
.create root\dlg=vs_dialog(_open=true)
.create root\dlg\table=vs_table()
.set root\\table._column_count=10
.set root\\table._row_count=10
root\\table._add_column_group(1,4)
root\\table._set_column_group_title(1,1,"Frozen Columns")
root\\table._freeze_columns(4)
```

9.10.12.2 Drawing border

VS_TABLE can be viewed with and without border. The viewing of borders is defined with the _HAS_BORDER attribute.

By default, this feature is enabled.

There is no GUI support for this attribute.

9.10.12.3 Grabbing hand

In VS_TABLE it is possible to have a grabbing feature on the table view at runtime.

The feature can be obtained using the _GRABBING_HAND_ENABLED attribute.

By default this feature is disabled. When the attribute is set to 1, the feature is available.

Once this feature is enabled, any cell in the table cannot be selected.

9.10.12.4 Sorting

Sorting is based on cell texts. If the cell does not contain any text, it is assumed that the text is an empty string. There are two possible ways to sort a table:

- Windows Explorer like behavior
- Defining _SORT_CRITERIA attribute

With the Windows NT Explorer like behavior, the programmer enables sorting for the table and for the column. The user can double-click the column title to sort the current column, first it will be in ascending order. If the user double-clicks again, it will be in descending order. This behavior affects the _SORT_CRITERIA attribute.

```
.set ._sorting_enabled=TRUE
._set_column_sorting(1,TRUE)
```

By defining the _SORT_CRITERIA attribute, the table has an attribute called _SORT_CRITERIA, which is an ordered vector containing vectors of column index and sort order. Defining the _SORT_CRITERIA will override previous sorting properties.

```
@criteriaVect=vector(vector(1,"ASCENDING"),-
vector(4,"ASCENDING"),vector(3,"DESCENDING"))
.set ._sort_criteria=%criteriaVect
._sort
```

A few points of interest demonstrated by the code above:

- line 1 defines a vector to be used as a sort criteria. In this case, the most significant key for sorting is column one, which will be sorted in ascending order. The second key for sorting is the column four, which will be sorted in ascending order too. And the third key for sorting is the column three, which will be sorted in descending order.
- line 3 executes the actual sorting

The previous code is shorter version of:

```
._set_column_sorting(1,TRUE)
._set_column_sorting(3,TRUE)
._set_column_sorting(4,TRUE)
@criteriaVect=vector(vector(1,"ASCENDING"),-
vector(4,"ASCENDING"),vector(3,"DESCENDING"))
.set ._sort_criteria=%criteriaVect
._sort
```

The sorting is not possible when some groups are defined.

9.10.12.5 Sorting Arrow

Sorting for VS_TABLE sorts the complete table based the column whose header is double clicked on and there will be UP/DOWN arrow image indication to denote whether table is sorted based on ascending/descending sort i.e. UP arrow will be drawn to indicate ascending sort and DOWN arrow to indicate descending sort. And this visibility of UP/DOWN arrow can be controlled by the attribute _SORT_ARROW_VISIBLE which can take input values of either TRUE or FALSE, setting this attribute TRUE will display the arrow image and setting the same to FALSE will not display any arrow image.

9.10.13 Value Handler

The value handler is what enables the cells of the VS_TABLE object to be edited. It is serving as the foundation for user defined value handlers. Message flows are described in Section 9.10.14.

9.10.13.1 Action Methods

For Visual SCIL Objects there are several action methods and action attributes for handling communication between the VS_TABLE and the user defined edit object or view object. There is a need for these action methods since the VS_TABLE handles only text strings (see _GET_CELL_TEXT_RANGE, _GET_CELL_TEXT, _SET_CELL_TEXT_RANGE and _SET_CELL_TEXT). The table needs text strings and that is why there must be a way to communicate with more complicated value handlers.

9.10.13.2 GET_TABLE_EDIT_TEXT

This action method is executed whenever the table needs a value from an edit object, and before the edit object of the cell is closed, the CELL_EDITING_ACCEPTED action method. This function can be used, for example, to convert an internal edit object value to text value, which is stored in the cell. GET_TABLE_EDIT_TEXT is called with two parameters: ROW and COLUMN of the current cell. This action method should return a text string.

9.10.13.3 GET_TABLE_VIEW_TEXT

This action method is executed whenever the table needs the value for the view object of the current cell, for example when the view object is drawn into the cell. This function can be used, for example, to convert edit object value of the cell into the text value of the cell.

9.10.13.4 UPDATE_TABLE_EDIT_TEXT

This action method is executed whenever the edit object of the cell needs a value to be shown, and before the editing of the cell is started (after the CELL_EDITING_STARTED action method). This action method has three parameters: ROW and COLUMN of the cell and TEXT of the cell. This action method could be used to convert the text value of the view object into the edit object value.

9.10.13.5 UPDATE_TABLE_VIEW_TEXT

This action method is executed whenever the table needs a value for the view object, and before the view object of the cell is drawn. This function can be used, for example, to convert the text value of the cell into the current value of the view object. This action method has three parameters: ROW, COLUMN and TEXT of the cell.

9.10.13.6 Action Attributes

9.10.13.7 _GET_TABLE_EDIT_TEXT

Setting of this attribute specifies the action method GET_TABLE_EDIT_TEXT.

9.10.13.8 _GET_TABLE_VIEW_TEXT

Setting of this attribute specifies the action method GET_TABLE_VIEW_TEXT.

9.10.13.9 _UPDATE_TABLE_EDIT_TEXT

Setting of this attribute specifies the action method UPDATE_TABLE_EDIT_TEXT.

9.10.13.10 _UPDATE_VIEW_TEXT

Setting of this attribute specifies the action method UPDATE_TABLE_VIEW_TEXT.

9.10.13.11 Restrictions

The following objects can be used as edit or view objects:

- VS_BOX
- VS_BUTTON
- VS_CHECK_BOX
- VS_COMBO_POPDOWN
- VS_CONTAINER
- VS_ICON_VIEW
- VS_LIST
- VS_NUMERIC_SPINNER
- VS_OPTION_BUTTON
- VS_PALETTE
- VS_TEXT
- VS_TEXT_SPINNER

Known limitations:

- The cell should not contain an object that has a scrollbar enabled, because the scrollbar may be drawn incorrectly (VS_COMBO_POPDOWN can be used because it may have a scrollbar on its pop-up window rather than on the parent table space.)
- Setting, for example colors, fonts and geometry for a view or an edit object is ignored. The table will set them, for example _SET_COLUMN_BACKGROUND_COLOR.
- Edit and view objects must be created or loaded as childs of an ancestor object of the table in the objects tree.
- If a sufficient time is passed between creating/loading the object and setting it as value handler, the object should be made invisible and disabled (if it is not expected to appear on the dialog and take mouse and keyboard events).
- One object can be set as edit or view object for only one cell, row or column.
- After the object has been set as value handler it cannot be deleted by SCIL .DELETE command before the table is deleted.

If a big table would be used defining a value handler for every cell would use a lot of memory. So it is more preferable to use _SET_COLUMN_VALUE_HANDLER or _SET_ROW_VALUE_HANDLER, if possible. In this case, the table will use only one value handler for the whole column or row.

9.10.13.12 Examples

9.10.13.13 Example 1

This example uses a VS_COMBO_POPDOWN as an edit item for the column 1. The view object can be NULL. In this case, the table needs the text string to be saved into the cell. The programmer must define the action methods for the edit object.

```
.load root\a=vs_combo_popdown(._source_file_name,"edit1_object")
mytable._set_column_value_handler(1, "", "root\a")
```

Code for GET_TABLE_EDIT_TEXT:

```
#if ._selected_index==0 #then #return ""
#else #return ._text
```

Code for SET_TABLE_EDIT_TEXT:

```
@txt_cell=argument(3)
#if %txt_cell=="" #then .set ._selected_index=1
#else .set ._text=%txt_cell
```

9.10.13.14 Example 2

To show an image in the cell, define a VS_BOX as a view object and for example a VS_COMBO_POPDOWN, VS_TEXT or VS_TEXT_SPINNER as an edit object.

```
.load root\viewObjcet=vs_box(._source_file_name,"view_object")
.load root\editObject=vs_text_spinner(._source_file_name,- "edit_object")
root\editObject._set_text_at(1,"Image1")
root\editObject._set_text_at(2,"Image2")
root\editObject._set_text_at(3,"Image3")
mytable._set_column_value_handler(2,"root\viewObject",-"root\editObject")
.load root\image1=vs_image(._source_file_name,"image1")
.load root\image2=vs_image(._source_file_namme,"image2")
.load root\image3=vs_image(._source_file_name,"image3")
```

Code for VS_TEXT_SPINNER action method GET_TABLE_EDIT_TEXT:

```
#return ._text
```

Code for VS_TEXT_SPINNER action method SET_TABLE_EDIT_TEXT:

```
@text=argument(3)
#if %text=="" #then ._select_text_at(1)
#else ._select_text(%text)
```

Code for VS_BOX action method UPDATE_TABLE_EDIT_TEXT:

```
@imageName=argument(3)
#if %text<>"" #then #block
.set ._image="root\\"+%imageName
#block_end
#else .set ._image="root\\image1"
```

9.10.14 Message Flows

It is appropriate to understand the sequence of execution of the action methods. In the following figures, the UML 1.1 notation for sequence diagrams has been used to describe the message flows. In message names, SCIL has been used as a programming language.

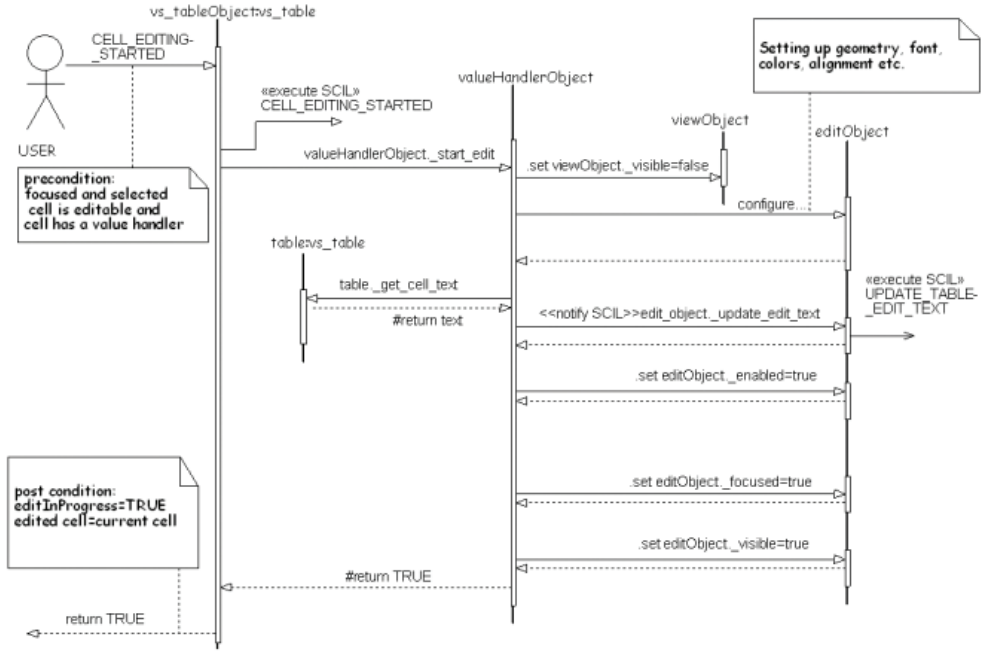


Figure 102: The user is starting the editing of the cell

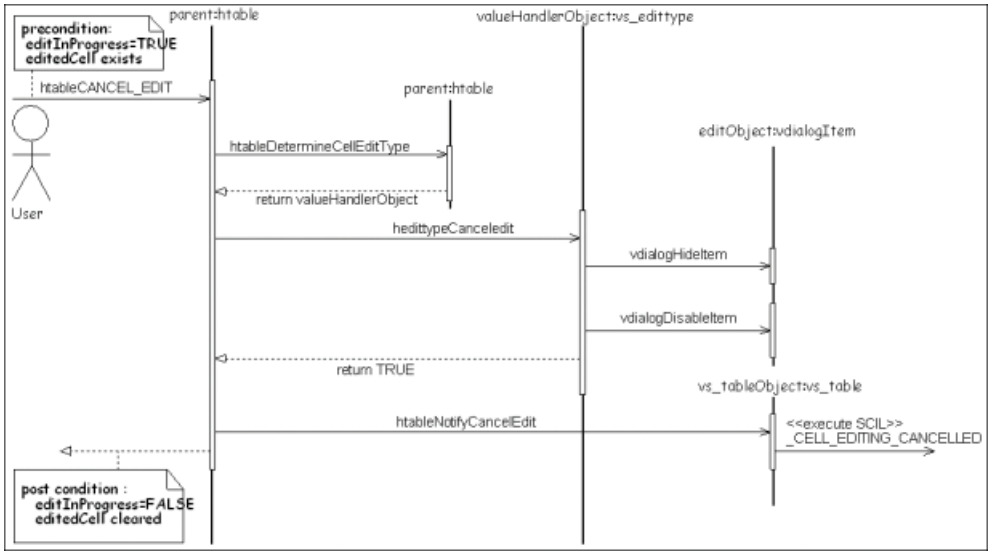


Figure 103: The user has cancelled the editing of the cell

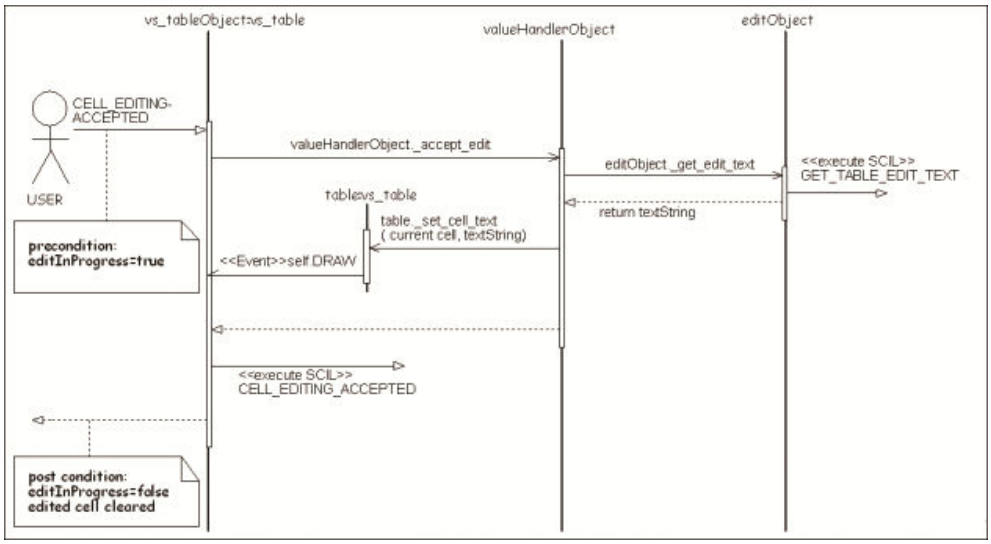


Figure 104: The user has accepted a new value for the cell

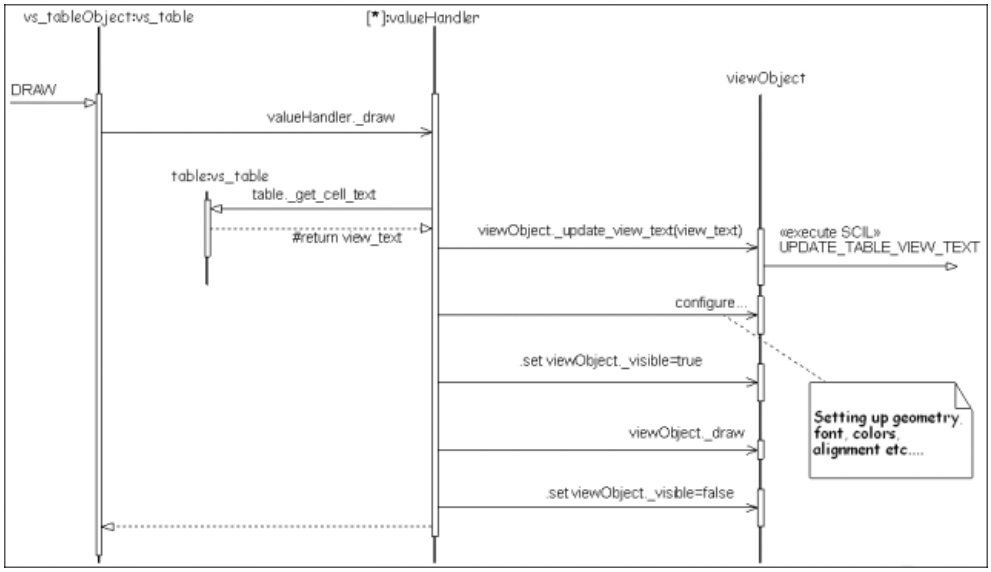


Figure 105: Redrawing the VS_TABLE

9.10.14.1 Selection Handling

Message order is different depending on whether the selection is done with mouse or keyboard. The programmer could ignore calls for _SELECTION_HAS_CHANGED, if the argument value is 1 or 2.

Example code for _SELECTION_HAS_CHANGED action method:

```
@arg=argument(1)
    #if %arg==3 #then #block
    ; in this case the selection has changed
    ; and it can be checked with _SELECTED_CELLS
    #block_end
```

A case when a cell is selected with keyboard:

- FOCUSED_CELL_CHANGED (cell to be focused)
- SELECTION_HAS_CHANGED (1)
- SELECTION_HAS_CHANGED(3)

A case when only one cell is selected with mouse:

- SELECTION_HAS_CHANGED (1)
- FOCUSED_CELL_CHANGED(cell to be focused)
- SELECTION_HAS_CHANGED(3)

A case when multiple cells are selected with mouse:

- _SELECTION_HAS_CHANGED (1)
- Multiple times _SELECTION_HAS_CHANGED(2)
- _FOCUSED_CELL_ CHANGED(cell to be focused)
- _SELECTION_HAS_CHANGED(3)

9.10.15 SCIL Exceptions

Table 2 and Table 3 contain some conditions that generate SCIL exceptions.

Table 2: Attributes and their conditions that generate SCIL exceptions

Attribute name	Condition
_COLUMN_COUNT	When setting: column count>=0 and column count<= maximum column count.
_FOCUSED_CELL	When setting: given row and column has valid values.
_ROW_COUNT	When setting: row count>=0 and row count<=maximum row count.
_SORTING_ENABLED	When enabling: row count >0.

Table 3: Methods and their conditions that generate SCIL exceptions

Method name	Condition
_SET_CELL_VALUE_HANDLER	Given row and column has valid value, view object and edit object are right type of Visual SCIL object.
_SET_COLUMN_VALUE_HANDLER	Given column has valid value, view object and edit object are right type of Visual SCIL object.
_SET_ROW_VALUE_HANDLER	Given row has valid value, view object and edit object are right type of Visual SCIL object.
_SET_CELL_TEXT _SET_CELL_TEXT_RANGE	New row or column count exceeds the maximum column or row count.
All other _GET_CELL and _SET_CELL methods	Given row and column has valid values.
All other _GET_COLUMN and _SET_COLUMN methods	Given column has valid value.
All other _GET_ROW and _SET_ROW methods	Given row has valid value.
_FREEZE_COLUMNS	Frozen column range does not occupy entire group.
_ADD_COLUMN_GROUP _ADD_ROW_GROUP	Incorrect group parameters: start or end column, overlapping groups.
_ADD_COLUMNS	Incorrect start number or column count.
_ADD_ROWS	Incorrect start number or row count.
_DELETE_COLUMNS	Incorrect start or end number.
Table continues on next page	

Method name	Condition
_DELETE_ROWS	Incorrect start or end number.
All _GET_COLUMN_GROUP and _SET_COLUMN_GROUP methods	Column group not found.
All _GET_ROW_GROUP and _SET_ROW_GROUP methods	Row group not found.

Section 10 Image Editor

10.1 About this Section

This section describes how to use the Image Editor, for example how to add or edit an image for an icon or a cursor.

10.2 General

Images can be drawn as image objects and stored in a file. They can also be drawn directly on an object (boxes, containers and buttons). Images can be added for all types of dialog items. They are an important part of image domain, icon view and cursor items. The user can also add an object of type Image in the Object List. In the Image Editor the user can edit, add, export or import images. Supported image formats are GIF, PGM, PPM, PBM, ICO, DIB and BMP.

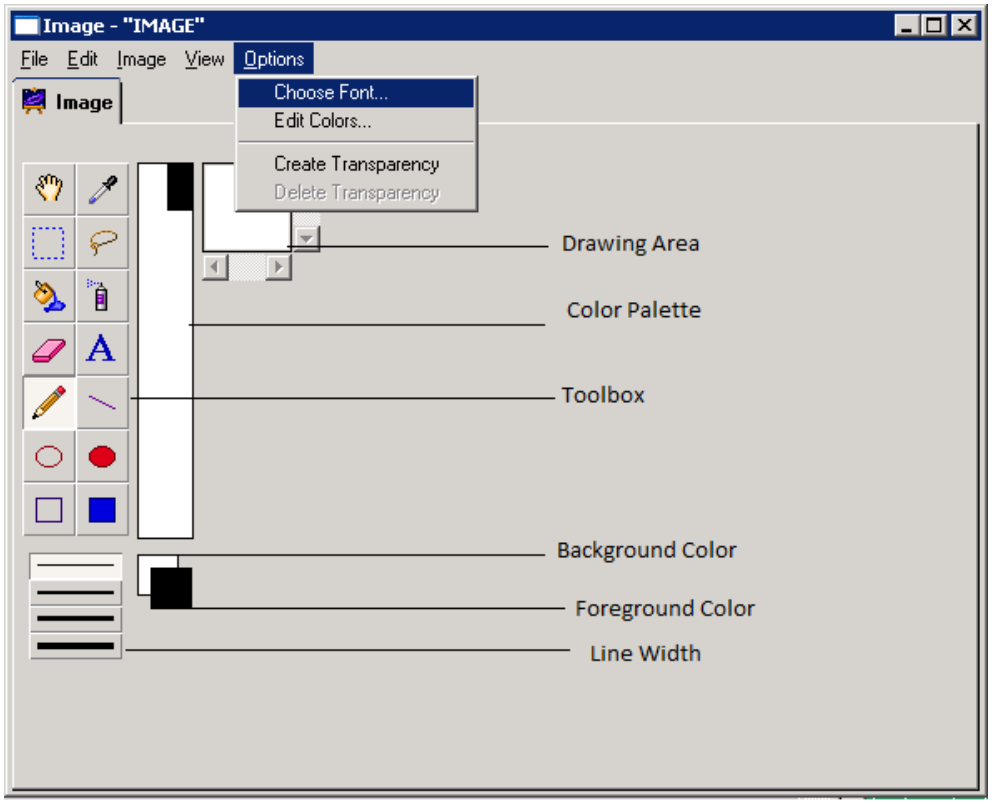


Figure 106: The Image Editor

10.3 Size of the Drawing Area

When an object of type Image has been added and entered into Image Editor, the drawing area is very small. The default height and width are ten pixels.

To resize the drawing area:

1. Choose **Set Size** from the Image menu.
2. In the dialog box that appears type the width and height you want to use.
3. Click **OK**.

10.4 Drawing an Image

Add and edit images with the toolbox tools. To draw an image:

1. Click the desired tool to be used.
2. Point to the drawing area.
3. Click or drag the mouse to make the desired image.

10.5 Toolbox

The tools are described in the same order as they appear in the toolbox.

Hand: Moves the visible part of the image when an oversize image is in the drawing area. Select and drag the oversize image with the Hand Tool to its new position.

Dropper: Replaces the foreground color with a color that is selected from the image. Click any color in the drawing area to pick up that color. The selected color is displayed in the Color Palette and the Foreground Color Box. To pick up a background color, hold down the CTRL key while clicking the Dropper.

Selection: Selects a rectangular area. Select the area by clicking the Selection tool and holding the mouse button down while dragging the pointer from one corner of the area to the opposite corner. To select the whole drawing area, double-click the tool.

Lasso: Selects an area, which does not have to be rectangular. Choose the area by clicking the Lasso tool and then holding the mouse button down while dragging the pointer from the beginning point of the area to the end point.

Fill: Fills an area with the Foreground color.

Spray Can: Draws with the Foreground color using a spray pattern.

Eraser: Erases the image. Drag the eraser over the area you want to erase. Press the SHIFT key before erasing to constrain the tool horizontally or vertically. To erase the whole drawing area, click the drawing area and then double-click the eraser.

Text: Allows the user to type text and place it in the image. Click the Text tool and then the drawing area. The dialog box shown in Figure 107 appears. Type the text and click **OK**. Once **OK** is clicked, the text cannot be edited. To change the text, delete the old text and type the new text from scratch. The text entered earlier can be removed using Edit/Undo menu command. The font style can be changed by choosing Set Font from the Tool menu. For more information on using the Font Chooser, see Section 7.

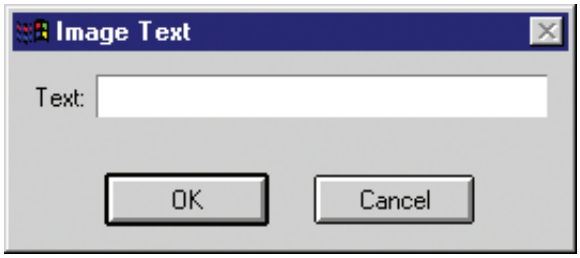


Figure 107: In this dialog box type the text you want to insert

Pencil: Draws one pixel at the time, using the Foreground color by default.

Line: Draws straight lines. Hold the SHIFT key down during the dragging to constrain the line horizontally, vertically or to a 45-degree angle.

Ellipse: Draws an ellipse. Hold the SHIFT key down during dragging to form a circle.

Filled Ellipse: Draws an ellipse filled with foreground color. Hold the SHIFT key down during dragging to form a filled circle.

Rectangle: Draws a rectangle. Hold SHIFT down during dragging to form a square.

Filled Rectangle: Draws a rectangle filled with the Foreground color. Hold the SHIFT key down during dragging to form a filled square.

10.6 Editing an Image

To use special edit functions:

1. Select the area to be edited with the Selection Tool or Lasso Tool.
2. Choose the desired edit function. Move the area by selecting the Hand tool or copy by using **Cut**, **Copy** and **Paste** functions from the Edit menu. Change the orientation of the area by using **Rotate**, **Flip Horizontal** or **Flip Vertical** functions. Rotation rotates the selection 90 degrees. **Flip Horizontal** mirrors the area horizontally and **Flip Vertical** mirrors the area vertically. It is also possible to limit the image to a selected area by using the Crop to Selection function. Choose these functions from the Edit menu.

10.7 Preview of an Image

To see the preview of the image, choose **Preview** from the View menu. The image is displayed as the users see it. The effect of changes as they are made can also be seen.

10.8 Saving Images

When the Image Editor is closed, all changes are propagated to the image. Abandon changes by using the **Edit/Undo** menu command or by closing the edited VSO file without saving.

10.9 Importing and Exporting Images

Image formats that are supported are GIF, PGM, PPM, PBM, ICO, DIB and BMP. To import an image:

1. Choose **Import** from the Image menu. The Open dialog box appears.
2. Select the correct file name extension for the image format being imported. The **Open** dialog box lists only directories and files with the specified file name extension.
3. Select the file name of the image being imported.
4. Click **OK**. The image appears in the drawing area. Imported images can be edited in the Image Editor. If the entire image is not visible, choose **Preview** from the View menu.

To export an image:

1. Choose **Export** from the Image menu. A dialog box appears.
2. Enter the filename and location for the image. The default format is GIF, which can also be changed. Click **OK**.

10.10 Color Setting

The Foreground and Background colors can be selected in several ways. One way of doing it is as follows:

1. To select the Foreground color, click a color in the Color Palette. The color is displayed in the Foreground Color Box.
2. To select the Background color, hold the CRTL key down while clicking a color in the Color Palette. The color is displayed in the Background Color Box.

The colors in the Color Palette are edited in the Color Palette Editor. To change the Color Palette:

1. Open the Color Palette Editor by choosing **Edit Colors** from the Options menu. The editor shown in Figure 108 appears with the current Color Palette.
2. Click one of the default color palettes from the **Palettes** menu. The default palettes are rainbow, pastels, earthtones, hues16 and EGA palettes.
3. Click **Install**. The chosen color palette is used in the Image Editor, which appears on screen again.

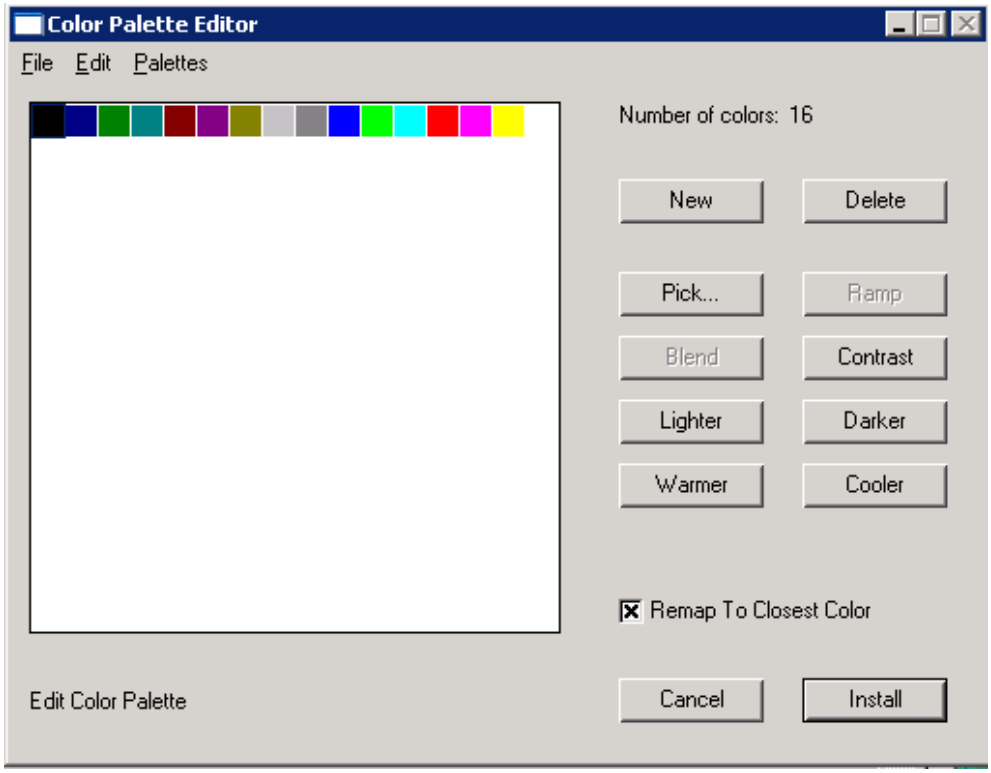


Figure 108: The Color Palette Editor, where you can choose the Color Palette you want to use in the Image Editor.

10.11 Line Width Setting

The line width that is used for drawing is chosen from the Line Width Palette. Line Tool, Ellipse Tool and Rectangle Tool use the line width chosen in the palette. To select a line width, click it in the palette.

10.12 Image Transparency

A transparency is a mask of an image. The areas of a transparency that are not white are masked out in the image.

To add a transparency for an image:

1. Choose **Add Transparency** from the Options Menu. The transparency editing area appears.
2. Select the area of the original image to be masked.
3. Choose **Mask Selection** from the Edit menu. The chosen area is shown masked on the transparency editing area. The mask can be edited. Note that the Foreground and Background colors cannot be changed, they are always black and white.

To remove transparency, choose **Delete Transparency** from the **Options** menu.

10.13 The Cursor Editor

The Cursor Editor is used to add cursors for dialog items. The cursor is drawn in the left drawing area. Use the center drawing area to define the hot spot for the cursor. The hot spot is reshown as a specially colored part. Use the drawing area on the right to add a mask for the cursor. A mask must be specified to make a cursor visible in an application.

From the Cursor menu, choose **Try Cursor**. A copy of your cursor appears. Choose **Try Cursor** again to return to the normal cursor.

Index

Hitachi ABB Power Grids
Grid Automation Products
PL 688
65101 Vaasa, Finland

https://hitachiabb-powergrids.com/microscadax

Scan this QR code to visit our website

1MRK 511 483-UEN