AppBuilder
By Magic Software Enterprises

Magic Software AppBuilder

Version 3.2

Getting Started Guide

Corporate Headquarters:

Magic Software Enterprises
5 Haplada Street,
Or Yehuda 60218, Israel
Tel +972 3 5389213
Fax +972 3 5389333

AppBuilder Solutions has made every effort to ensure that the information contained in this document is accurate; however, there are no representations or warranties regarding this information, including warranties of merchantability or fitness for a particular purpose. AppBuilder Solutions assumes no responsibility for errors or omissions that may occur in this document. The information in this document is subject to change without prior notice and does not represent a commitment by AppBuilder Solutions or its representatives.

Getting Started Guide

Introduction to Getting Started Guide

Introduction

AppBuilder is a suite of tools used for developing, building, and testing software applications rapidly and efficiently. These tools include the following:

- Diagrammers for design activities
- Tools to create the user interface and the high-level code itself
- Debuggers for running an application in test mode
- Wizards and tutorials to help you develop applications faster

This guide teaches you how to use many of these tools by building a sample application. Steps in this tutorial include building the application hierarchy, creating the user interface, creating and preparing rules, and configuring for deployment and execution. It takes about three hours to complete the tasks and create the project.

Intended Audience

This book is the right starting place if you are new to the AppBuilder environment. Some basic application development experience helps, but is not required. You should be familiar with Microsoft Windows.
This guide is part of the AppBuilder documentation set. Other manuals contain more detailed procedures and more complete reference material. See AppBuilder Documentation for more information on the documentation package.

AppBuilder Documentation

The documentation package for AppBuilder includes a complete online help system and a set of the manuals in PDF format. The online help system contains the same information that the manuals contain and can be accessed directly from the application, using the *Help* menu or *F1* . The manuals in PDF format are available on the AppBuilder Installation CD and on the AppBuilder Customer Service web site (see Getting Additional Assistance for detailed information).

Documentation Conventions

This book uses the conventions described in the following table:

Conventions used in AppBuilder documentation

This Convention	(Example)	Specifies this type of information
Courier font	*DEV_GRT_LB*	Type the text (commands, code, etc.) exactly as it appears in this guide.
Boldface	*OK*	Select the *option* , *function* , *button* , or *selection* in the window.
Menu choices	*Analysis > Verify*	Select the menu and option.
Key combination	*Alt + B*	Press the specified keys, at the same time.
Numbered procedure, in sequence	1. Type the name. 2. Select *OK* . 3. Close the window.	Perform the specified steps, in order.
Bulleted list	• Apples • Oranges • Pears	Select from a list of valid options.

Numbered procedure or task list, with options.	1. Open the file: a. Double-click the icon. b. Select *Open* . The Open window displays. 2. Select the choice, then select *OK* .	Perform a high-level task or procedure. The procedure might include sub-tasks.
File names, variable names, and file paths.	start <*filename*>	Replace the italic text with a site-specific variable or option.

The following table summarizes the product documentation:

Documentation

Introductory Documents	**Workstation Operation Documents**	**Reference Documents**
Getting Started Guide	*Developing Applications Guide*	*Development Tools Reference Guide*
Installation Guide for Windows	*Debugging Applications Guide*	*Scripting Tools Reference Guide*
Communications Guide	*Deploying Applications Guide*	*ObjectSpeak Reference Guide*
IVP User Guide	*Reports Guide*	*Rules Language Reference Guide*
	Repository Administration Guide for Workgroup and Personal Repositories	*System Components Reference Guide*
	Multi-Language User Interface Guide	*Messages Reference Guide*
		INI Settings Reference Guide

Descriptions of each guide are provided in the following sections.

Introductory Documents

Table 1-3 summarizes the documentation containing instructions about installing the product as well as other introductory information.

Introductory documents summary

Title	Description
Getting Started Guide	Explains the product documentation and provides a brief tutorial that builds a simple application to help you familiarize yourself with the Construction Workbench interface. Steps in this tutorial include building the application hierarchy, creating the user interface, and creating and preparing rules.
Installation Guide for Windows	Provides installation and configuration instructions for various parts of the AppBuilder product including workstation Construction Workbench, workgroup repository, and workgroup servers on Windows platforms.
Communications Guide	Provides the procedures for configuring the internal communications and setting up servers and machines from the workstation.
IVP User Guide	Provides a series of tests to verify that AppBuilder and the third-party applications it needs for your development environment are set up and configured properly. IVP also serves as another sample application written with AppBuilder.

Workstation Operation Documents

The following table summarizes the documents that contain information about standard application operation. Some documents contain information about interactions between workstations and servers or mainframes, but most relate to workstation operation.

Workstation operation documents summary

Title	Description
Developing Applications Guide	Provides the procedures for using AppBuilder to create applications. Includes information about project development, rules, preparation, and deployment. Describes both stand-alone and distributed applications in multiple environments.
Debugging Applications Guide	Provides the procedures for debugging and troubleshooting the application.

Deploying Applications Guide	Provides the procedures for configuring, packaging, deploying, and executing applications on distributed machines in the network. Includes steps for configuring the execution environments and details about web archive deployment and remote rule execution.
Reports Guide	Provides the procedures for creating reports on the workstation with Report Painter and seeing the results on the mainframe with Enterprise Report Writer or on web servers with Java Report Writer.
Repository Administration Guide for Workgroup and Personal Repositories	Provides detailed information about the use of Windows-based development workstation repositories, including configuration, migration, and systems administration.
Multi-Language User Interface Guide	Explains the multi-language user interface and how to use it to develop applications with language-dependent user interfaces that can be targeted to different language environments.

Reference Documents

The following table summarizes the documents that contain general AppBuilder product reference information.

Reference documents summary

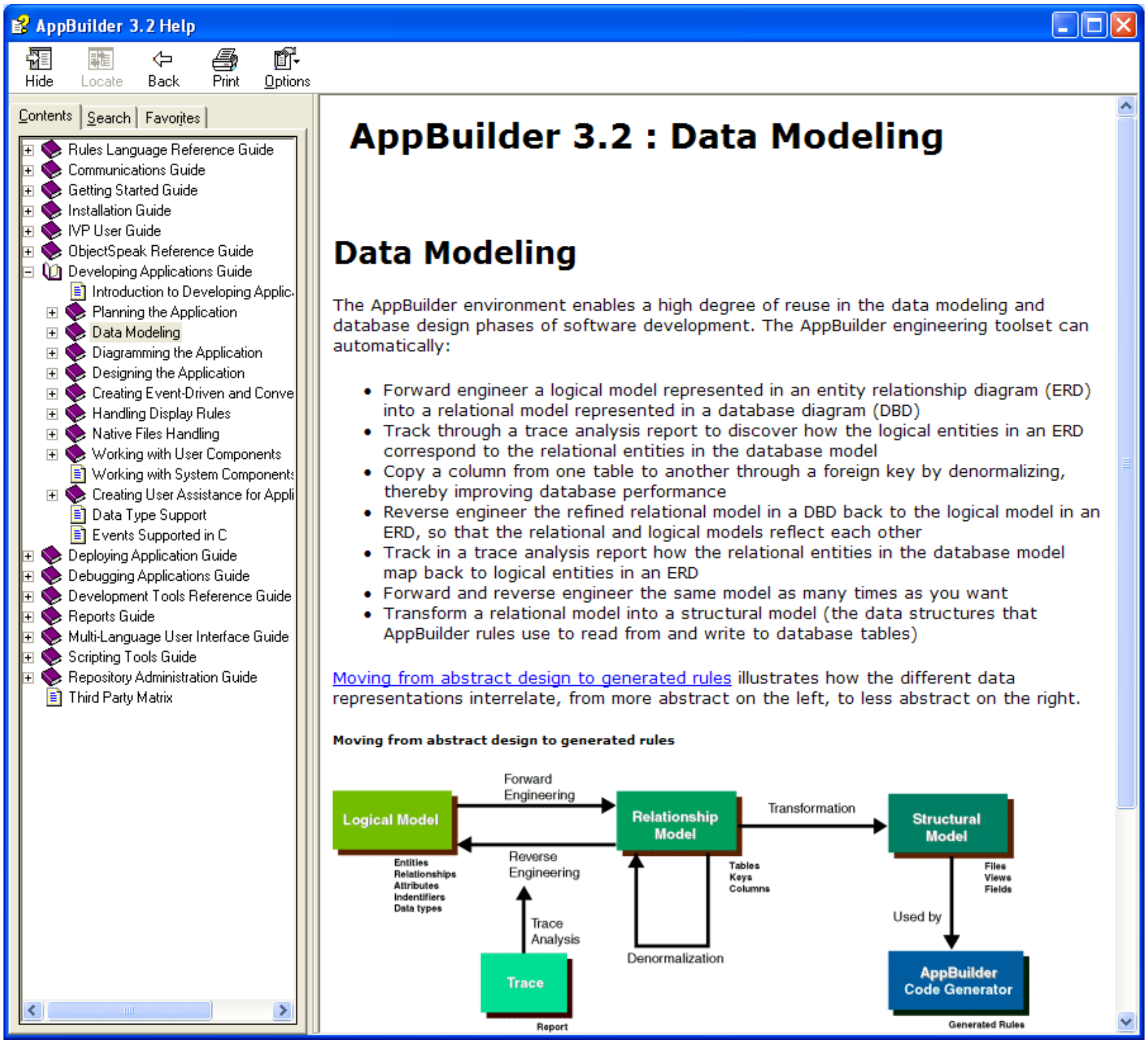
Title	Description
Development Tools Reference Guide	Provides a reference for the many tools available in the Construction Workbench for developing applications. Includes a summary of the toolbars, the Hierarchy window tabs, the Window Painter, the Menu Editor, the Rule Painter, and other tools for planning and developing applications.
Scripting Tools Reference Guide	Describes the TurboScripter capabilities and offers a complete reference for TurboCycler object generation and task automation capabilities. Includes a description of the Developer Kit's open architecture for automatically generating repository objects and the ActiveX object that provides a scriptable repository interface.
ObjectSpeak Reference Guide	Lists the object dot notation that is available for use with Rules Language rules in application development. This version with Java support is called ObjectSpeak.
Rules Language Reference Guide	Describes the AppBuilder Rules Language used to specify the processing logic of the application and how the entities that comprise an application interact.
System Components Reference Guide	Lists the available system components that can be included in applications development.
Messages Reference Guide	Lists the preparation time and runtime information and error messages.
INI Settings Reference Guide	Lists the settings in the initialization files that affect the behavior and operation of AppBuilder.

Using the Online Help

Use the online help on the workstation to view detailed information about working in the AppBuilder environment. From the desktop, select *Start > All Programs > AppBuilder > Documentation > AppBuilder32.chm*. To access the online help while using the application, press *F1* or select *Help > Contents* from the Construction Workbench menu bar.
The online help window includes three panes. The top pane contains toolbar controls for the window. The navigation pane provides multiple means to access the contents of the help. The content pane provides the topics themselves.

- Toolbar
- Navigation Pane
- Content Pane

Online help example



- Rules Language Reference Guide
- Communications Guide
- Getting Started Guide
- Installation Guide
- IVP User Guide
- ObjectSpeak Reference Guide
- Developing Applications Guide
  - Introduction to Developing Applic
  - Planning the Application
  - Data Modeling
  - Diagramming the Application
  - Designing the Application
  - Creating Event-Driven and Conve
  - Handling Display Rules
  - Native Files Handling
  - Working with User Components
  - Working with System Components
  - Creating User Assistance for Appli
  - Data Type Support
  - Events Supported in C
- Deploying Application Guide
- Debugging Applications Guide
- Development Tools Reference Guide
- Reports Guide
- Multi-Language User Interface Guide
- Scripting Tools Guide
- Repository Administration Guide
- Third Party Matrix

AppBuilder 3.2 : Data Modeling

Data Modeling

The AppBuilder environment enables a high degree of reuse in the data modeling and database design phases of software development. The AppBuilder engineering toolset can automatically:

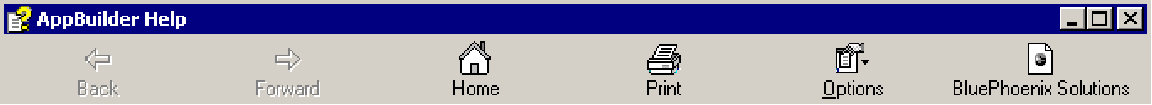
- Forward engineer a logical model represented in an entity relationship diagram (ERD) into a relational model represented in a database diagram (DBD)
- Track through a trace analysis report to discover how the logical entities in an ERD correspond to the relational entities in the database model
- Copy a column from one table to another through a foreign key by denormalizing, thereby improving database performance
- Reverse engineer the refined relational model in a DBD back to the logical model in an ERD, so that the relational and logical models reflect each other
- Track in a trace analysis report how the relational entities in the database model map back to logical entities in an ERD
- Forward and reverse engineer the same model as many times as you want
- Transform a relational model into a structural model (the data structures that AppBuilder rules use to read from and write to database tables)

Moving from abstract design to generated rules illustrates how the different data representations interrelate, from more abstract on the left, to less abstract on the right.

Moving from abstract design to generated rules

Toolbar

AppBuilder Help Toolbar



The toolbar buttons on the online help window provide the following options:

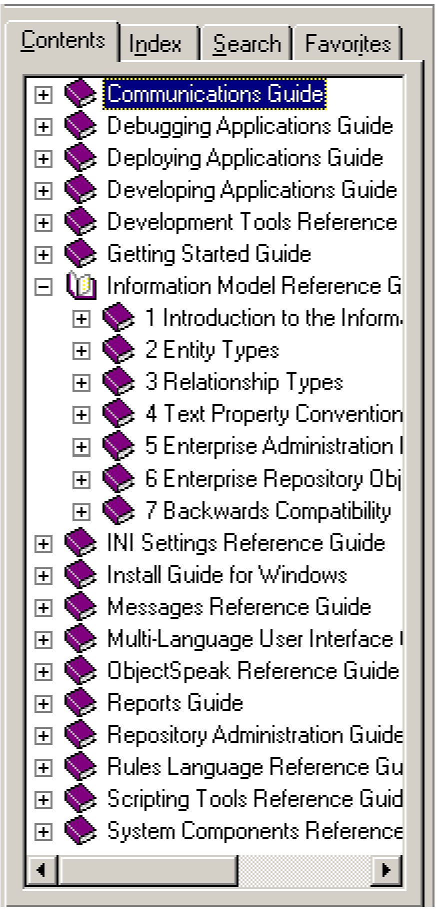
Help toolbar buttons

Button	Function
Back	Moves backward in the sequence of topics that have been displayed in the window.
Forward	Moves forward in the sequence of topics that have already been displayed in the window.
Home	The default screen for the help system.
Print	Prints either the designated topic or that topic and all subtopics to a specified printer.
Options	Provides multiple options for the window display, including hiding the navigation tab, changing browser options, and others.

Navigation Pane

Use the Navigation Pane in any of several ways to find the information you need more quickly than by paging through AppBuilder help.

Navigation Pane



- Communications Guide
- Debugging Applications Guide
- Deploying Applications Guide
- Developing Applications Guide
- Development Tools Reference
- Getting Started Guide
- Information Model Reference G
  - 1 Introduction to the Inform
  - 2 Entity Types
  - 3 Relationship Types
  - 4 Text Property Convention
  - 5 Enterprise Administration I
  - 6 Enterprise Repository Obj
  - 7 Backwards Compatibility
- INI Settings Reference Guide
- Install Guide for Windows
- Messages Reference Guide
- Multi-Language User Interface I
- ObjectSpeak Reference Guide
- Reports Guide
- Repository Administration Guide
- Rules Language Reference Gu
- Scripting Tools Reference Guid
- System Components Reference

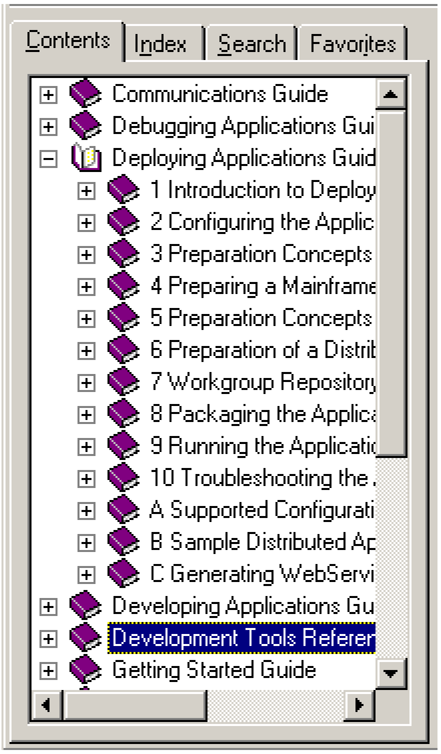
You can use the navigation pane in any of the following ways:

- Using the Contents Tab
- Using the Index Tab
- Using the Search Tab
- Using the Favorites Tab

Using the Contents Tab

Click the *Contents* tab to display a table of contents for all of the guides, including all of their chapters and sections. These contents contain all major topics. The contents tab normally does not display pages with examples as well as pages with detailed lists.

Contents tab



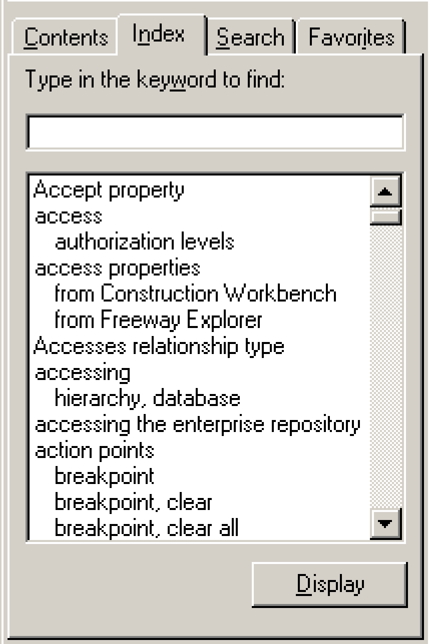
- Communications Guide
- Debugging Applications Gui
- Deploying Applications Guid
  - 1 Introduction to Deploy
  - 2 Configuring the Applic
  - 3 Preparation Concepts
  - 4 Preparing a Mainframe
  - 5 Preparation Concepts
  - 6 Preparation of a Distrit
  - 7 Workgroup Repository
  - 8 Packaging the Applica
  - 9 Running the Applicatio
  - 10 Troubleshooting the .
  - A Supported Configurati
  - B Sample Distributed Ap
  - C Generating WebServi
- Developing Applications Gu
- Development Tools Referen
- Getting Started Guide

Double-click a book icon. This opens the book and displays the topics beneath it. Continue to double-click to navigate to the topic you want to view. Click the topic icon ? to display the selected help text. The contents of the topic display in the content pane. To collapse or expand a topic, click the plus or minus signs or double-click an icon.

Using the Index Tab

Click the *Index* tab to display a list of indexed terms included in the online help. The index includes topics from the entire documentation set.

Index tab

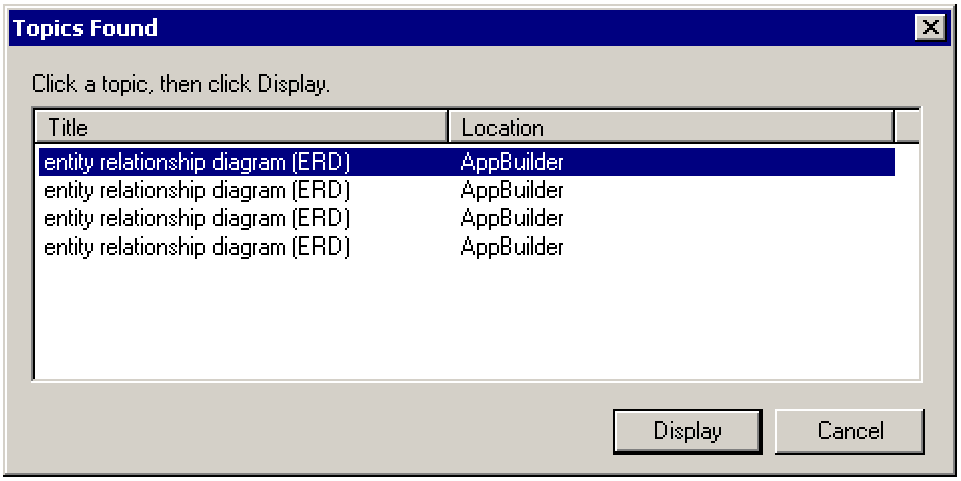


Type in the keyword to find:

Accept property
access
  authorization levels
access properties
  from Construction Workbench
  from Freeway Explorer
Accesses relationship type
accessing
  hierarchy, database
accessing the enterprise repository
action points
  breakpoint
  breakpoint, clear
  breakpoint, clear all

Select an item or topic of interest and click *Display* to view the related help text.
If the index term appears in more than one topic, the system displays a list of the books and links to topics. Select a topic by clicking the topic title. An example is shown in the following figure:

Multiple topics shown in content pane



Topics Found

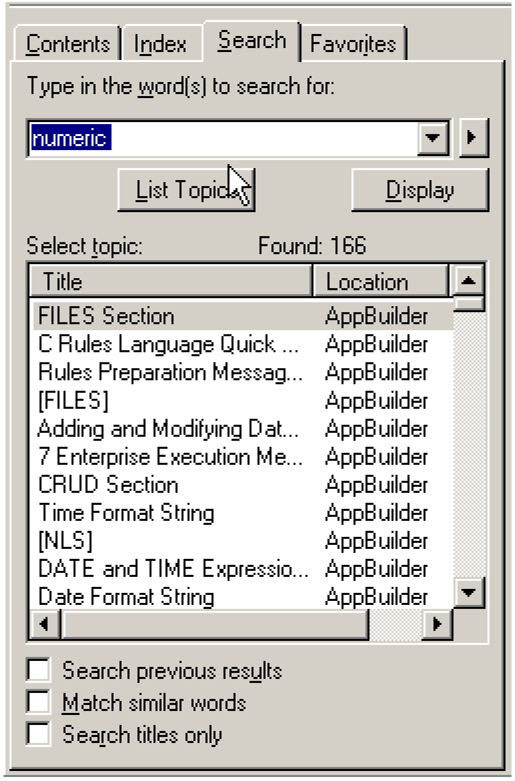
Click a topic, then click Display.

Title	Location
entity relationship diagram (ERD)	AppBuilder
entity relationship diagram (ERD)	AppBuilder
entity relationship diagram (ERD)	AppBuilder
entity relationship diagram (ERD)	AppBuilder

Using the Search Tab

Click the *Search* tab to search the online help for information related to a key word or words. Type any key word or words and click *Display* .

Search tab



Type in the word(s) to search for:

numeric

Select topic: Found: 166

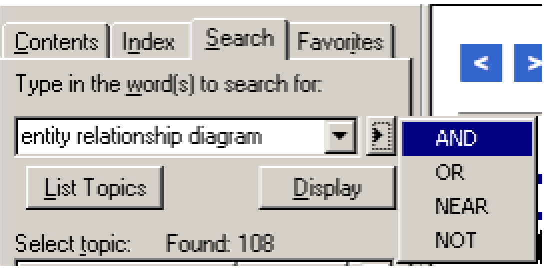
Title	Location
FILES Section	AppBuilder
C Rules Language Quick ...	AppBuilder
Rules Preparation Messag...	AppBuilder
[FILES]	AppBuilder
Adding and Modifying Dat...	AppBuilder
7 Enterprise Execution Me...	AppBuilder
CRUD Section	AppBuilder
Time Format String	AppBuilder
[NLS]	AppBuilder
DATE and TIME Expressio...	AppBuilder
Date Format String	AppBuilder

□ Search previous results
□ Match similar words
□ Search titles only

The results are listed by topic title. Double-click any search result to display the selected topic in the content pane.
Some important items to consider when searching:

- The search engine does not provide any feedback as a word is typed. No results are displayed until the *List Topics* button is clicked.
- The search engine allows for boolean searches. Click the

▶ button to the right of the search field to search on a combination of terms:

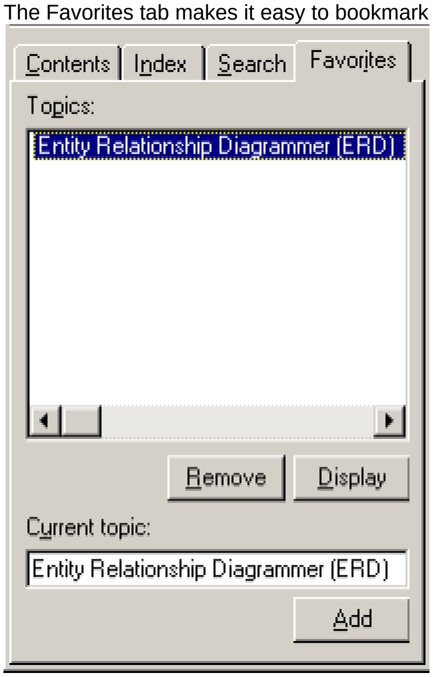


Use a boolean search to narrow the search.

- If you type multiple search words, the help system searches for topics that contain *all* of the search words.
- The help system "remembers" your previous searches. To recall previous searches, select the *Search previous results* check box.
- Other options allow for matching similar words and searching only topic titles.

Using the Favorites Tab

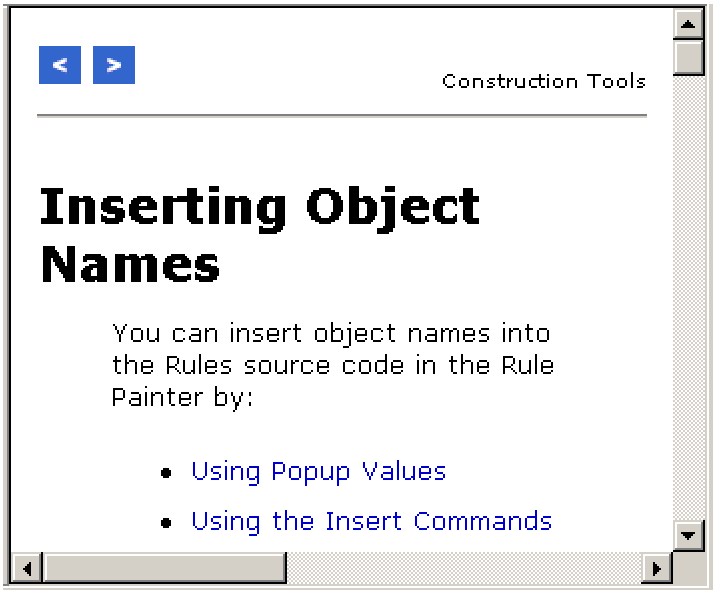
The Favorites tab makes it easy to bookmark a particular topic and recall it quickly. Add topics using the *Add* button.


Contents
Index
Search
Favorites
Topics:
Entity Relationship Diagrammer (ERD)
Remove
Display
Current topic:
Entity Relationship Diagrammer (ERD)
Add

Content Pane

The Content pane displays the AppBuilder help content. From this pane you can read the content, follow links to related or referenced topics, or move within the help content by going to the previous or next topic.

Sample Help Text pane



< >

Construction Tools

Inserting Object Names

You can insert object names into the Rules source code in the Rule Painter by:

- Using Popup Values
- Using the Insert Commands

Use the *Previous* < and *Next* > buttons to navigate throughout the help text. These buttons in the online help system move sequentially through the topics in the manual. They are separate and different from the toolbar's Forward and Back buttons.
Some text and graphics contain hyperlinks in blue text. Click the link to display the associated information.

Using the Printable Manuals

The documentation set also contains manuals in a format which is easily printable---PDF format (portable document format files created using Adobe? Acrobat?). You can read and print the documentation using Adobe Acrobat. You can access the printable manuals from the AppBuilder Installation CD or you can search on the AppBuilder Customer Service web site (see also Getting Additional Assistance for further information) .
See AppBuilder Documentation for a summary of the books.
In the Printable Documents window, select a book from the list. This opens the book in Adobe Acrobat format. Page through the documentation or use the table of contents on the left for easy navigation. You can print the entire document or specific pages. For more information about using Adobe Acrobat, refer to the Adobe Acrobat documentation.

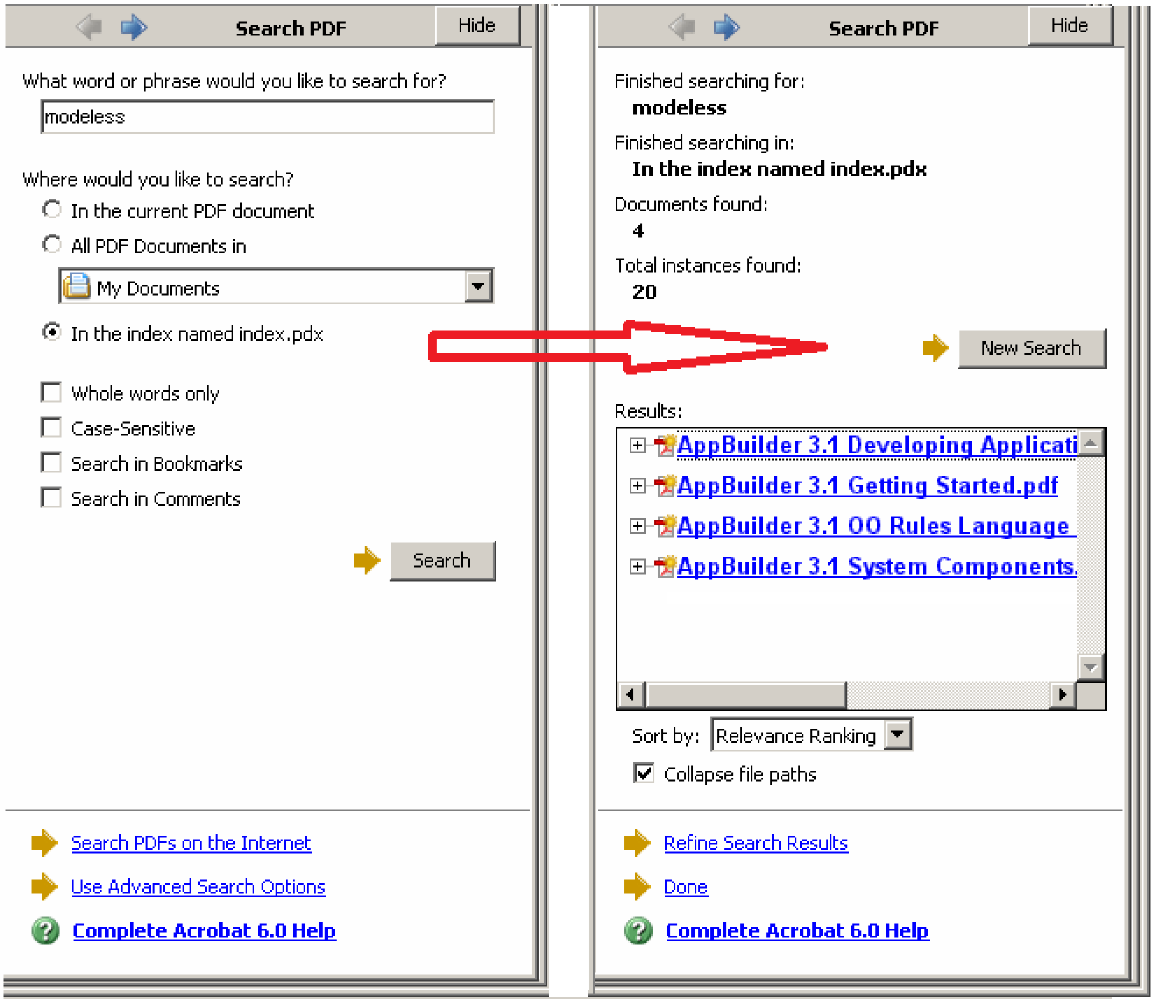
> ℹ Adobe Acrobat is required to search the master index and display and print documents in PDF format. You can download a free copy of Acrobat Reader from the Internet at http://www.adobe.com

The printable manuals include the documentation set for AppBuilder. The online and searchable version of these guides are also available by pressing the *Help* menu item or pressing *F1* from the AppBuilder Construction Workbench.

Searching the Printable Books

To search the entire set of AppBuilder books for specific information, open the printable book and click the Search button on the toolbar .
Use the *Search* button to perform a full text search of all the books.
In the Search dialog box (Example search of the printable books), type the word or words you want to search. If they only occur once in the set, the exact book and page are opened for viewing. If they occur more than once, a list of the books is displayed in a Search Results dialog. From that list, select the book for which you want to see the results. You can come back to this list of books at any time by clicking the Search Results button in the toolbar .

Example search of the printable books


Search PDF
Hide
Search
Search PDFs on the Internet
Use Advanced Search Options
Complete Acrobat 6.0 Help
Search PDF
Hide
New Search
Refine Search Results
Done
Complete Acrobat 6.0 Help

What word or phrase would you like to search for?

modeless

Where would you like to search?

○ In the current PDF document

○ All PDF Documents in

My Documents

● In the index named index.pdx

□ Whole words only

□ Case-Sensitive

□ Search in Bookmarks

□ Search in Comments

Finished searching for:
modeless

Finished searching in:
In the index named index.pdx

Documents found:
4

Total instances found:
20

Results:

- AppBuilder 3.1 Developing Applicati
- AppBuilder 3.1 Getting Started.pdf
- AppBuilder 3.1 OO Rules Language
- AppBuilder 3.1 System Components.

Sort by: Relevance Ranking

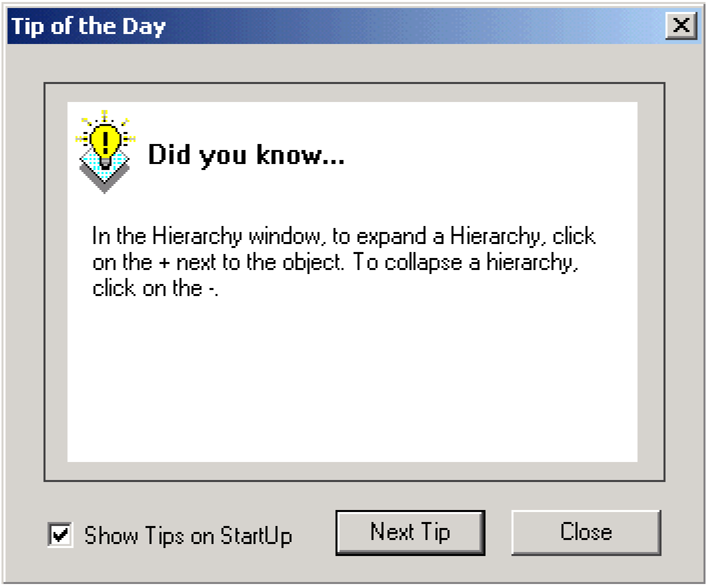
☑ Collapse file paths

You can use this search feature from any of the books.

Using the Tip of the Day

AppBuilder comes with tips that you can display about optimal use of the tools. When you first run the product, the tip of the day dialog is automatically displayed, as shown in Tip of the Day example. You can also display the tips by selecting *Help > Tip of the Day* from the Construction Workbench menu bar. Select the check box if you want the Tip of the Day to show at each startup. Click *Next Tip* repeatedly to look at several of the tips sequentially.

Tip of the Day example



In the Hierarchy window, to expand a Hierarchy, click on the + next to the object. To collapse a hierarchy, click on the -.

Finding Version Information

To find out which version of AppBuilder components, programs, or fixes you have installed on a machine, select from the Construction Workbench Help > About Magic Software AppBuilder > AppBuilder Info.

Getting Additional Assistance

After you have reviewed the product documentation, either Using the Online Help or Using the Printable Manuals, and you still have a question or require some assistance, check with your system administrator for problems relating to the installation or network operation of the product.
If you still need assistance or for the latest documentation and updates about the product, contact AppBuilder Customer Support using the information given in this section.
Contact AppBuilder Customer Support:

- Internet: http://support.appbuilder.com
- E-mail: *AppBuilderSupport@magicsoftware.com*

Starting the Integrated Environment

AppBuilder provides a robust, integrated toolset to design, implement, and maintain high volume, multi-platform, distributed applications. The first step is to start the environment on your workstation, which involves the following:

- Connecting to a Repository
- Starting AppBuilder
- Understanding the Construction Workbench
- Planning the Sample Application

Connecting to a Repository

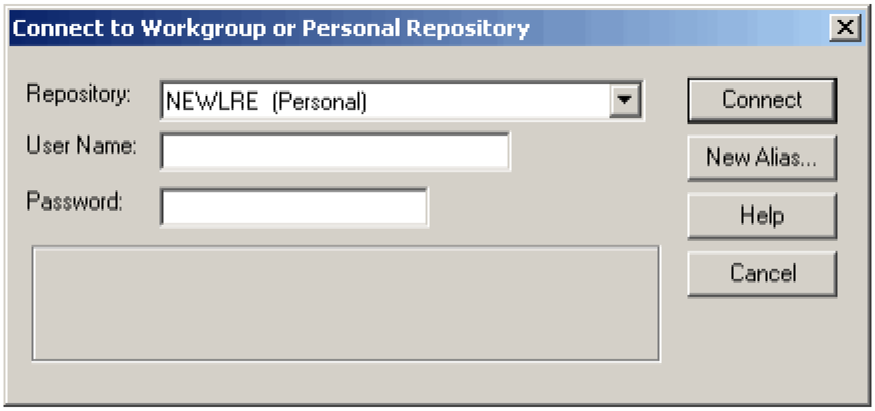
When starting AppBuilder, you must access the repository that you installed when you installed AppBuilder (refer to *Installation Guide for Windows*) to store your project and options. A repository is an intelligent, relational database that holds software application objects, the methods and facilities used to access the objects and the relationships among the objects.
There are three types of repositories:

- Personal (workstation)
- Workgroup (department server)
- Enterprise (mainframe)

For more information on the installation options for repositories, see the *Installation Guide for Windows* . For details about these repositories, how to install them, how to connect to them, and how to maintain them, refer to the *Repository Administration Guide for Workgroup and Personal Repositories* .
This exercise is specific to building an application using a Personal repository.

1. From the Windows desktop, click ***Start* ,** then select ***Programs > AppBuilder > Construction Workbench***.
   The Connect to Repository dialog displays, as shown below:

Connect to Repository dialog



Repository: NEWLRE (Personal)
User Name:
Password:

2. From the *Repository* dropdown list, select the repository to access.
3. Type your *User name* and *Password* and click *Connect* .
   The AppBuilder Construction Workbench displays. Construction Workbench window identifies the parts of the workbench. The system also displays the Tip of the Day.
4. Click *Close* to close the Tip of the Day window and begin using AppBuilder.

> If the Construction Workbench takes a long time to load or the repository takes a long time to find, you might have the ODBC trace set on your workstation. If the ODBC trace is set, a lot of logging is being done when accessing the underlying repository database. Once the tracing is stopped and this setting is applied, the connection time returns to normal. In *Control Panel > Administrative Tool > Data Sources (ODBC) > Tracing* , set *Stop Tracing Now* and click *OK* .

Starting AppBuilder

To start AppBuilder from the Start menu, select AppBuilder from the Start menu. A submenu displays with a menu of choices:

- Configuration
- Documentation
- Execution Clients
- Repository
- Construction Workbench

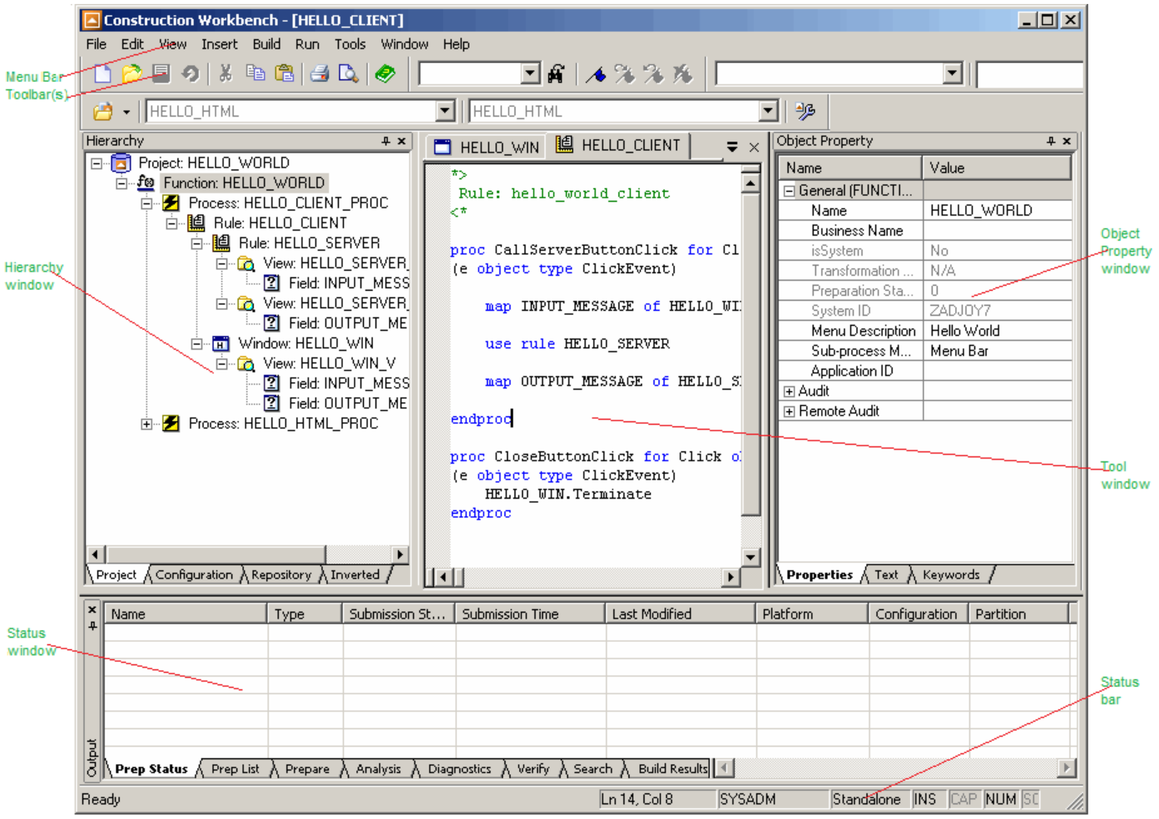
You can perform many AppBuilder configuration tasks by selecting the Configuration, Execution Clients, and Repository menu items. However, you will perform most of your development in AppBuilder using the Construction Workbench.

Understanding the Construction Workbench

The Construction Workbench (see Construction Workbench window) contains all the tools needed to build the application hierarchy. The main parts of the Construction Workbench are:

- Menu bar ? Provides access to AppBuilder commands
- Toolbar ? Includes a collection of buttons that provide quick access to tools, commands, and objects
- Hierarchy window ? Contains tabs to build project hierarchy and deployment configurations and provides an area to store and query objects from the repository and display the parents of an object
- Status window ? Contains tabs to monitor a process such as preparation, analysis, or debugging
- Status bar ? Displays the current preparation mode, help for each tool, and other information
- Tool window ? Provides a work area for the specific AppBuilder tools (for example, Window Painter or Rule Painter)
- Object Property window - Contains editable fields with informations about the object preferences.

Construction Workbench window



```
*>
 Rule: hello_world_client
<*

proc CallServerButtonClick for Cl
(e object type ClickEvent)

    map INPUT_MESSAGE of HELLO_WI

    use rule HELLO_SERVER

    map OUTPUT_MESSAGE of HELLO_S

endproc

proc CloseButtonClick for Click o
(e object type ClickEvent)
    HELLO_WIN.Terminate
endproc
```

Name	Value
⊟ General (FUNCTI...	
Name	HELLO_WORLD
Business Name	
isSystem	No
Transformation ...	N/A
Preparation Sta...	0
System ID	ZADJOY7
Menu Description	Hello World
Sub-process M...	Menu Bar
Application ID	
⊞ Audit	
⊞ Remote Audit	

Name	Type	Submission St...	Submission Time	Last Modified	Platform	Configuration	Partition

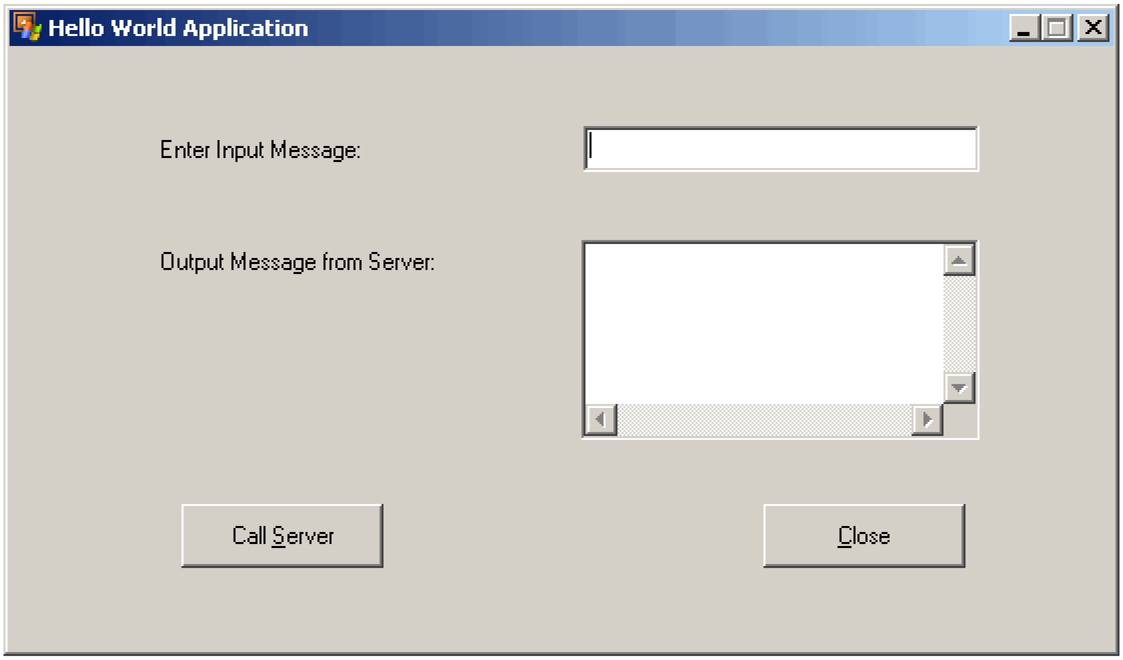
The AppBuilder Construction Workbench has the following features to help you work more efficiently:

- The system displays the name of any button or icon over which you move the cursor.
- Each part of the workbench (for example, Hierarchy or Status window) can be resized or converted into a floating window and positioned anywhere on the Windows desktop.
- The system only displays menus, toolbars, and options that are valid at any given time, based on the currently selected object.
- To customize the interface and tools, select *Tools* > *Workbench Options* from the Construction Workbench menu. Each tool within the Construction Workbench (for example, Hierarchy window or preparation tool) contains a tab with tool-specific options. These are explained more fully in the *Development Tools Reference Guide* .

Planning the Sample Application

This guide takes you through the process of building an application that echoes input data back to you in an output field (See Hello World application). By creating this sample application, you can learn to use the main AppBuilder tools to build the application hierarchy, create the user interface, create and prepare rules, and configure the application for deployment and execution.

Hello World application



The sample application is designed as a standalone Java client application that can run locally on your machine. [Creating the Configuration Hierarchy]() discusses other possible configuration and deployment scenarios.

> ⚠ You must have the Java runtime compiler installed on your workstation PC to create the sample application. Consult your system administrator for more information.

When building the sample application, *commit* your work to the repository often. Select **File > Commit** from the Construction Workbench menu, click the **Commit** button 💾 in the toolbar or press **Ctrl+M** on your keyboard to save the sample application. If the **Commit** button is dimmed, the repository already contains the most recent changes.

Defining the Overall Application Hierarchy

The hierarchy diagram describes the structure or part of the structure of an application. This structure specifies how the application executes. Use the *Project* tab of the Hierarchy window to build the objects, such as rules and windows, in the application hierarchy and their relationships. The tasks in defining the hierarchy include the following:

- Using the Hierarchy Toolbars
- Creating the Project
- Adding a Function
- Adding a Process
- Adding a Rule
- Adding a Window
- Adding Views
- Adding Fields

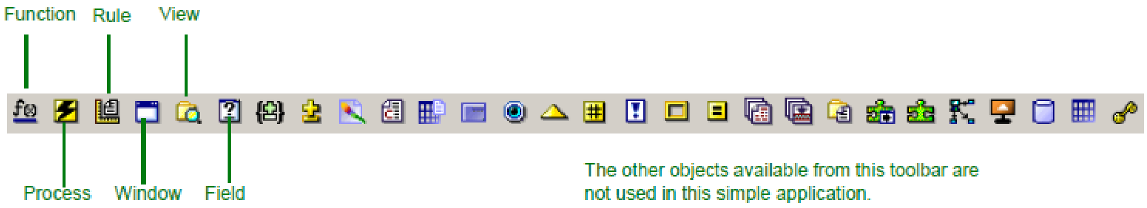
Using the Hierarchy Toolbars

AppBuilder contains two Hierarchy toolbars for manipulating objects in the project hierarchy.

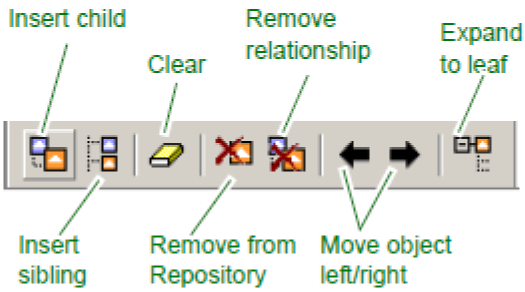
1. Open the Construction Workbench and log on to the Personal Repository. See Connecting to a Repository.
2. If the Hierarchy window is not displayed, select *View* > *Hierarchy* .
3. To add the hierarchy toolbars, select *View* > *Toolbars* > *Hierarchy* - *Objects* and *Hierarchy* - *Operations* from the Construction Workbench.

When you select each toolbar, it is displayed in the Construction Workbench.

Hierarchy - Objects toolbar (used with Hierarchy diagrammer)


Function
Rule
View
Process
Window
Field
The other objects available from this toolbar are not used in this simple application.

Hierarchy - Operations toolbar



In addition to using the toolbar, you can also use one of the following methods to work with objects:

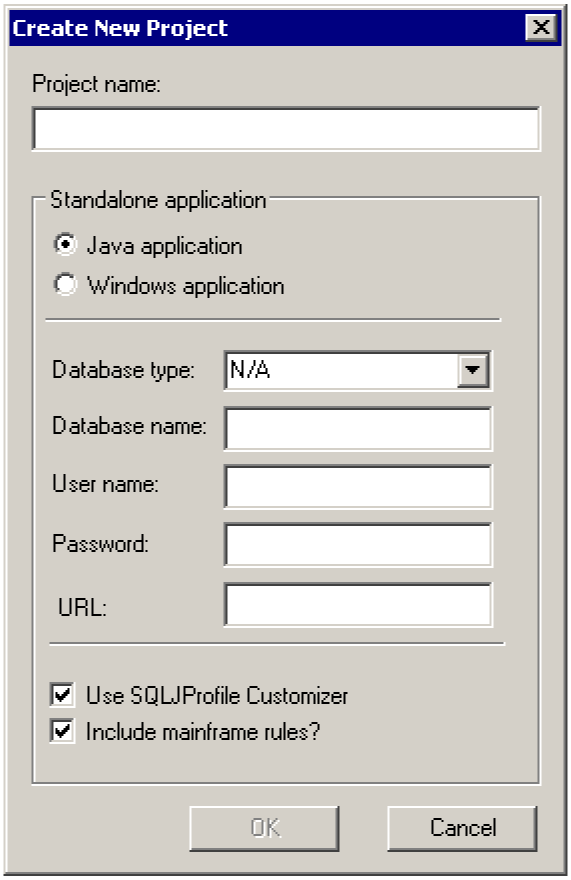
- Right-click an object in the Hierarchy window and select a function from the pop-up menu.
- Select an object in the Hierarchy window and select *Insert* or *Edit* from the Construction Workbench.

Creating the Project

In an AppBuilder application, the Project object contains the business function (see Adding a Function) and associated configurations of the application. The project is the highest level of the application hierarchy. For this sample application, we create the project illustrated in Planning the Sample Application.

1. Select ***File > New Project*** from the Construction Workbench menu. The Create New Project window displays, as shown in Create New Project window.

Create New Project window



Create New Project

Project name:

Standalone application

● Java application

○ Windows application

Database type: N/A

Database name:

User name:

Password:

URL:

☑ Use SQLJProfile Customizer

☑ Include mainframe rules?

2. Configure the following options for the sample application:

Field	**What to enter**	**Description**
Project Name	*HELLO_WORLD*	Name of the project. (For most of the objects you create, the name can be only 30 characters long, upper case, case insensitive, with no special characters or spaces; only underscore is allowed. The OO objects' names are mixed case, case sensitive, no special characters or spaces, underscore allowed. The Insert window does not allow any mistake, disabling the Insert button when you type a wrong character)
Standalone application	*Java application*	AppBuilder prepares the application and rules to execute in a Java environment.
Database type	N/A	The sample application runs locally on your PC and does not require a connection to a database or server so this setting does not matter. Leave the other database fields (*Database name* , *User name* , *Password and URL*) blank.
Use SQLJProfile Customizer	N/A	This utility augments the profile with DB2-specific information for use at run time.The SQLJ translator performs analysis on the SQLJ source file by checking for incorrect SQLJ syntax. For the sample application, this box does not need to be checked.
Include mainframe rules?	N/A	Any rules that would run on a mainframe would be prepared locally on your PC if this were checked. When not checked, it only checks syntax and does not prepare those rules. Since the sample contains none, the setting does not matter.

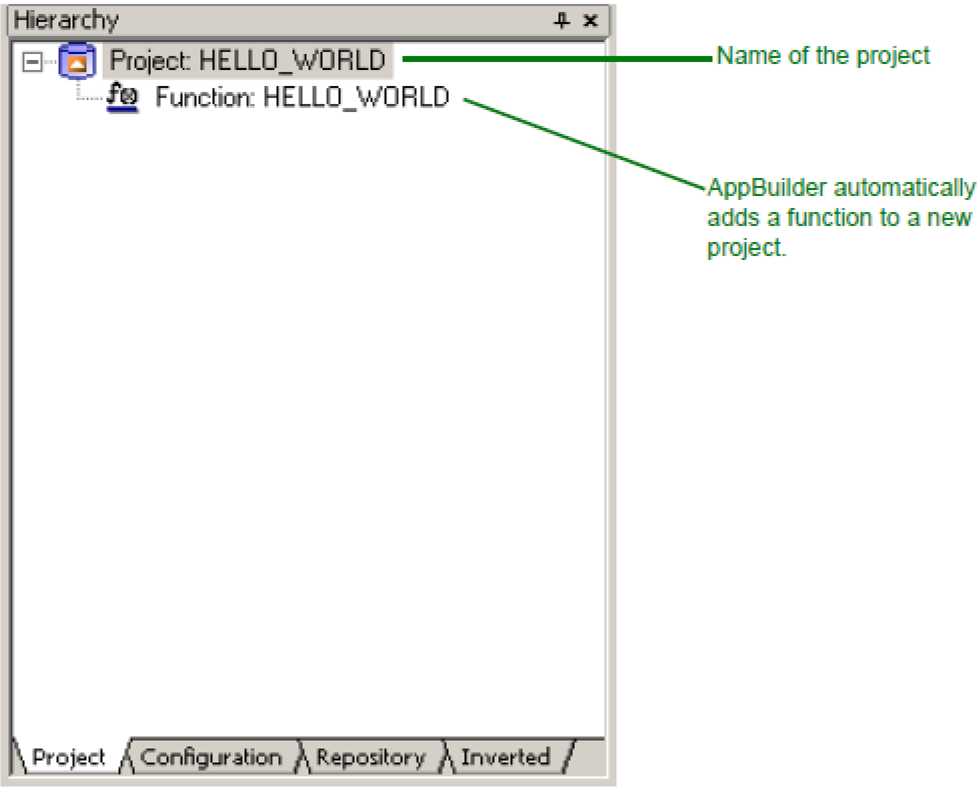
⛔ If you use a Workgroup or Enterprise repository, add a unique identifier to the end of each object you create for the sample application (for example, *Project Name*). This eliminates potential problems if other developers also attempt to create the sample application.

3. Click *OK* .

The Hierarchy window displays, as in Hierarchy window with function, showing the new project as the top of the application hierarchy on the *Project* tab. *Standalone* displays in the status bar of the Construction Workbench.

If the system displays a *Security Validation Failed* message contact your system administrator or repository administrator to gain the necessary permissions.

Hierarchy window with function



Adding a Function

A function represents the highest-level business function ? the purpose of the application. At execution time, the function may appear as a desktop icon or menu option that the end-user can access. Functions contain groups of processes that subdivide the application.

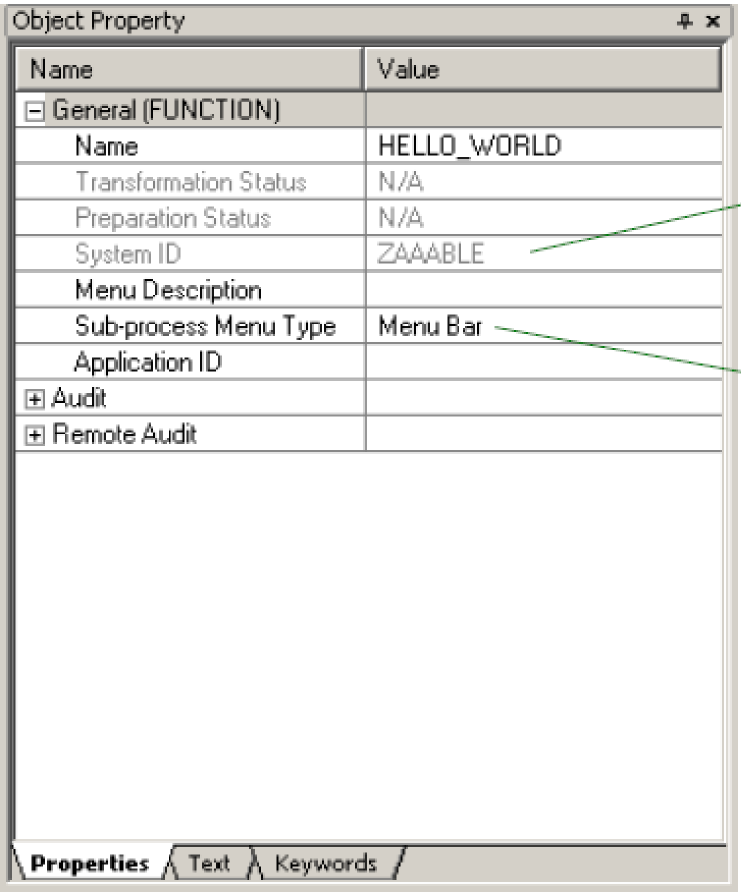
In this sample application, the function accepts user input, and then the input echoes back to the user.

1. Click the ***HELLO_WORLD*** function on the ***Project*** tab of the Hierarchy window. The Object ***Property*** window displays the properties, as shown in the following figure.

   You can configure the Construction Workbench options to use double-click to show the properties of an object from the Hierarchy window.

2. Type *Hello World Application* in the *Menu description* field and press Enter.

   Object Property dialog



Name	Value
⊟ General (FUNCTION)	
Name	HELLO_WORLD
Transformation Status	N/A
Preparation Status	N/A
System ID	ZAAABLE
Menu Description	
Sub-process Menu Type	Menu Bar
Application ID	
⊞ Audit	
⊞ Remote Audit	

The system automatically assigns a unique system ID for each object in the hierarchy.

Specifies how the functions under this process display to the end user.

Adding a Process

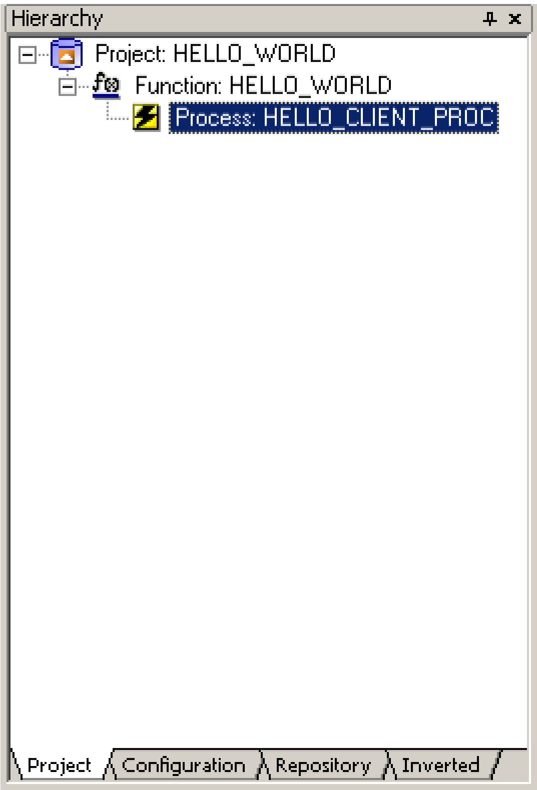
A process represents a logical unit of work or activity that can manipulate the data within the application. In an AppBuilder application, processes appear as options on pull-down menus.

For the sample application, we create a process to allow user input and system output.

1. Select the ***HELLO_WORLD*** function and click the ***Process*** button **in the Hierarchy Objects toolbar.**
   You can also add a process by:
   - Right-clicking the *HELLO_WORLD* function and selecting *Insert Child* > *Process* from the pop-up menu.
   - Selecting the *HELLO_WORLD* function and selecting *Insert Child* > *Process* from the Construction Workbench menu.
   - Clicking the *Insert Child* button
     in the Hierarchy Operations toolbar and selecting *Process* from the pop-up menu.
2. In the Insert Process dialog, type *HELLO_CLIENT_PROC* in the *Name* field and click *Insert* .

The Hierarchy Diagrammer shows the new process as a child of the *HELLO_WORLD* function, as shown in the following figure:

Hierarchy window with process



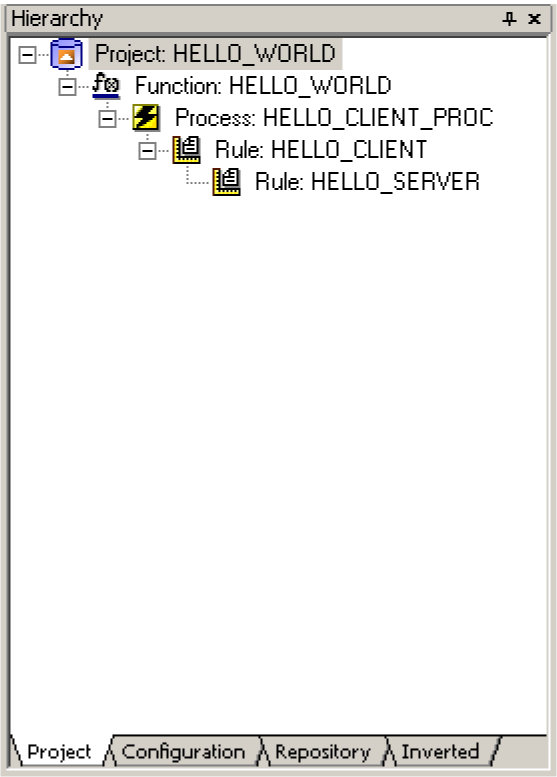
Adding a Rule

In an AppBuilder application, rules are programming statements that define the logic of the application (that is, how the application works). At execution time, each rule performs a unique action.
This sample application requires two rules: one rule (*client*) to handle the user interface (input and output) and one rule (*server*) to handle the business logic. In this section, we add the rules to the hierarchy. In Creating Rules we add the actual rule code.

1. Select the ***HELLO_CLIENT_PROC*** process and click the ***Rule*** button in the hierarchy toolbar.
   You can also add a rule by:
   - Right-clicking the *HELLO_CLIENT_PROC* process and selecting *Insert Child* > *Rule* from the pop-up menu.
   - Selecting the *HELLO_CLIENT_PROC* process and selecting *Insert Child* > *Rule* from the Construction Workbench menu.
   - Clicking the *Insert Child* button
     in the Hierarchy Operations toolbar and selecting *Rule* from the pop-up menu.
2. In the Insert Rule dialog, type *HELLO_CLIENT* in the *Name* field and click *Insert* . This rule accepts the user input and displays the output.
   The Hierarchy Diagrammer now shows the new rule as a child of the process.
3. Add a second rule as a child of the *HELLO_CLIENT* rule. Name this rule *HELLO_SERVER* . This rule copies the user input to the output.

Your hierarchy diagram should look like the one shown in the following figure:

Hierarchy window with rules



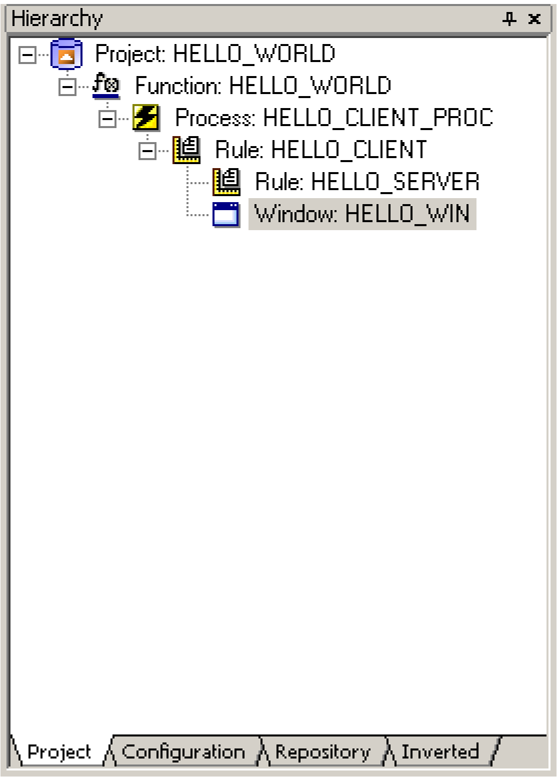
Adding a Window

Windows accept input or display data to the end user. In an AppBuilder application, windows are linked to *views* that contain the actual data elements, such as fields, for the window.
For our sample application, we need to create a window to handle the end-user input and the system output. In this procedure, we add the window to the hierarchy. In Creating the User Interface we create the actual window graphical user interface (GUI).

1. Select the ***HELLO_CLIENT*** rule and click the ***Window*** button in the hierarchy toolbar.
   You can also add a window by:
   - Right-clicking the *HELLO_CLIENT* rule and selecting *Insert Child* > *Window* from the pop-up menu.
   - Selecting the *HELLO_CLIENT* rule and selecting *Insert Child* > *Window* from the Construction Workbench menu.
   - Clicking the *Insert Child* button
     in the Hierarchy Operations toolbar and selecting *Window* from the pop-up menu.
2. In the Insert Window dialog, type *HELLO_WIN* in the *Name* field and click *Insert* .

The Hierarchy Diagrammer now shows the window as a child of the *HELLO_CLIENT* rule and a sibling of the *HELLO_SERVER* rule, as shown in the following figure:

Hierarchy window with window



Adding Views

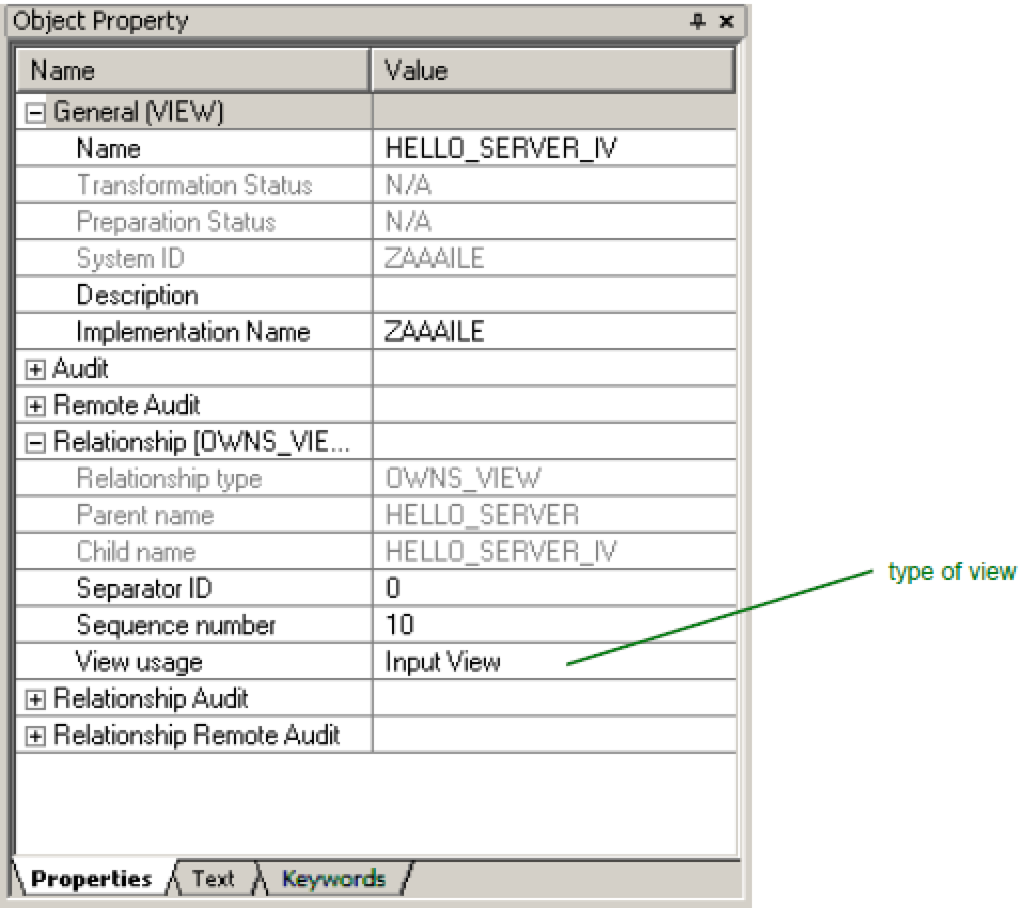
Views are structures that contain data elements, such as fields or other views, and define the data structure of the application. Views attached to a window are called *window message views* . These views define the data displayed to a user, data entered by a user, and data stored by the system.
For our sample application, we need to create views to handle input by the user, output to the user, and data displayed on a window. To create views, complete the following steps:

1. Select the ***HELLO_SERVER*** rule and click the ***View*** button in the hierarchy toolbar.
   You can also add a view by:
   - Right-clicking the *HELLO_SERVER* rule and selecting *Insert Child* > *View* from the pop-up menu.
   - Selecting the *HELLO_SERVER* rule and selecting *Insert Child* > *View* from the Construction Workbench menu.
   - Clicking the *Insert Child* button in the Hierarchy Operations toolbar and selecting *View* from the pop-up menu.
2. In the Insert View dialog, type *HELLO_SERVER_IV* in the *Name* field and click *Insert* .

   > The *IV* indicates that this is an *input* view. We recommend that you use a standard naming convention to easily identify the objects in your applications and repositories.

   The Hierarchy Diagrammer now shows the new view as a child of the rule.
3. Double-click the newly created view. The *Object Property* panel displays the information, as shown in the following figure:

   Object Property - Relationship [OWNS_VIEW] dialog



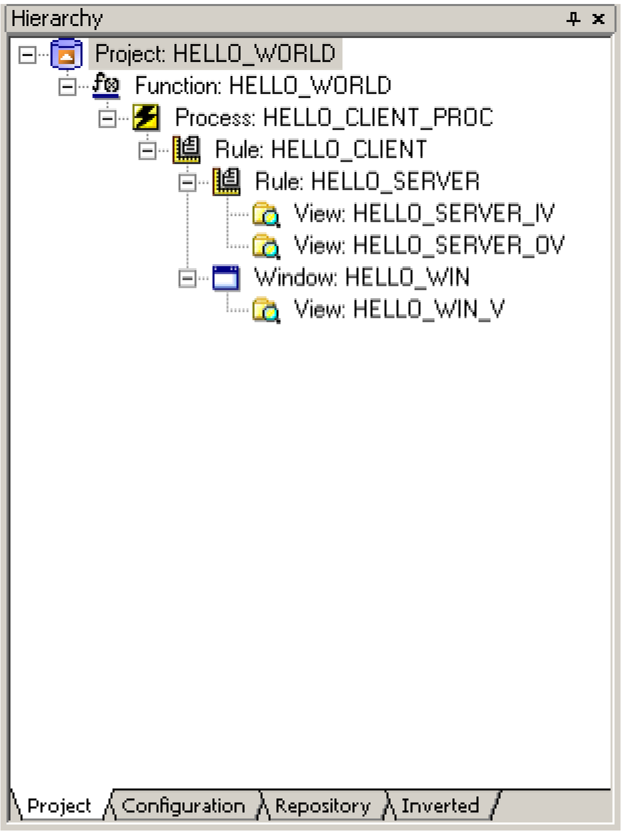
Name	Value
⊟ General (VIEW)	
Name	HELLO_SERVER_IV
Transformation Status	N/A
Preparation Status	N/A
System ID	ZAAAILE
Description	
Implementation Name	ZAAAILE
⊞ Audit	
⊞ Remote Audit	
⊟ Relationship [OWNS_VIE...	
Relationship type	OWNS_VIEW
Parent name	HELLO_SERVER
Child name	HELLO_SERVER_IV
Separator ID	0
Sequence number	10
View usage	Input View
⊞ Relationship Audit	
⊞ Relationship Remote Audit	

type of view

4. Click the Relationship[OWNS_VIEW]section to display the option fields, select *Input View* from the *View usage* drop-down list and press *Enter* . This specifies how the parent rule uses the data in this view.
5. Repeat steps 1 - 4. Replace steps 2. and 4. with the following additional views:
   - A view as a child of the *HELLO_SERVER* rule. Name this view *HELLO_SERVER_OV* . Set the *View usage* to *Output View* . This view handles the output view.
   - A view as a child of the *HELLO_WIN* window. Name this view *HELLO_WIN_V.* Set the *View usage* to *Input & Output View* . This view handles the end-user window.

Your hierarchy diagram now looks like the one shown in Hierarchy window with views.

Hierarchy window with views



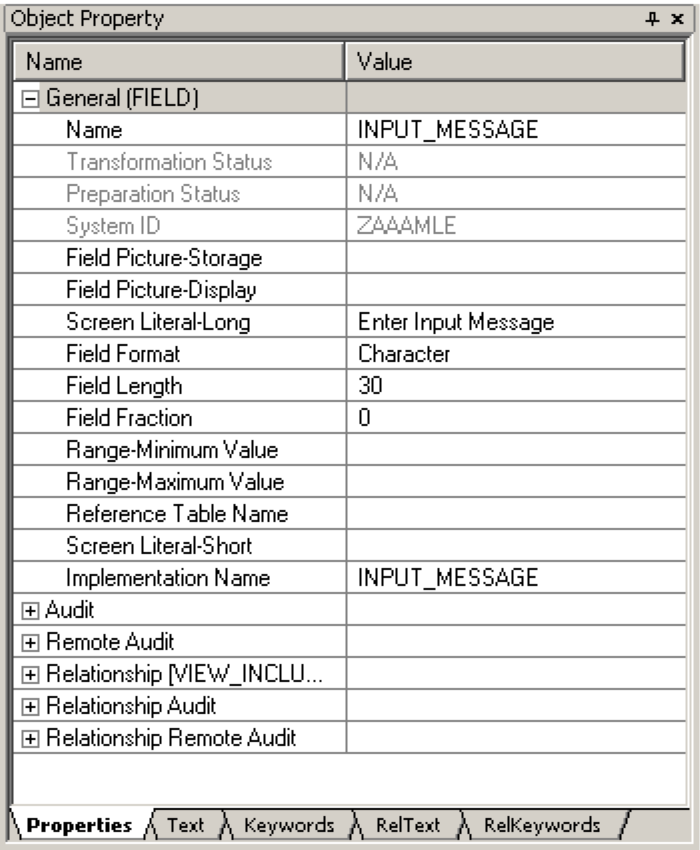
Adding Fields

Fields are the smallest unit of data in an AppBuilder application. Fields can record the input/output definition of objects or contain information about part of a file, such as a column in a database table. An edit field object defines a field in which the end user can either enter and change data, or view read-only data.
For the sample application, we need to create input and output fields for each view. To create input and output fields for each view, complete the following steps:

1. Select the ***HELLO_SERVER_IV*** view and click the ***Field*** button in the hierarchy toolbar.
   You can also add a field by:
   - Right-clicking the *HELLO_SERVER_IV* view and selecting *Insert Child* > *Field* from the pop-up menu.
   - Selecting the *HELLO_SERVER_IV* view and selecting *Insert Child* > *Field* from the Construction Workbench menu.
   - Clicking the *Insert Child* button in the Hierarchy Operations toolbar and selecting *Field* from the pop-up menu.
2. In the Insert Field dialog, type *INPUT_MESSAGE* in the *Name* field and click *Insert* . This field handles the end-user input.
   The Hierarchy Diagrammer now shows the new field as a child of the view.
3. Double-click the *INPUT_MESSAGE* field in the Hierarchy Diagram window and the Object Property window displays the information, as shown in Object Property - General [Field] dialog.

> You can configure the Construction Workbench options to use double-click to show the properties of an object from the Hierarchy window. Select *Tools* > *Workbench Options* from the Construction Workbench menu. Use the *Double-click actions* field on the *Hierarchy* tab to specify the default double-click action.

Object Property - General [Field] dialog



Name	Value
⊟ General (FIELD)	
Name	INPUT_MESSAGE
Transformation Status	N/A
Preparation Status	N/A
System ID	ZAAAMLE
Field Picture-Storage	
Field Picture-Display	
Screen Literal-Long	Enter Input Message
Field Format	Character
Field Length	30
Field Fraction	0
Range-Minimum Value	
Range-Maximum Value	
Reference Table Name	
Screen Literal-Short	
Implementation Name	INPUT_MESSAGE
⊞ Audit	
⊞ Remote Audit	
⊞ Relationship [VIEW_INCLU...	
⊞ Relationship Audit	
⊞ Relationship Remote Audit	

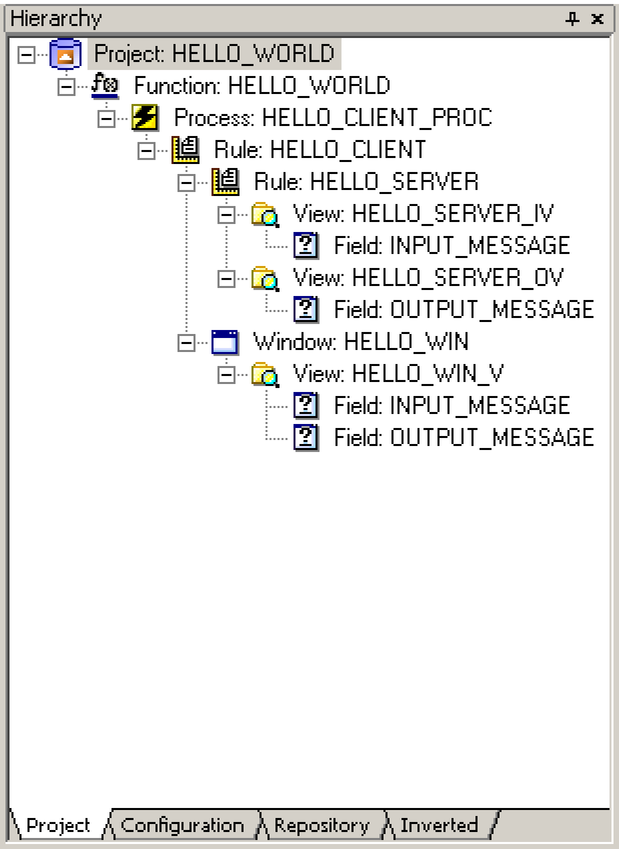
1. Enter or change the following fields:
   - *Screen literal-long* : *Enter Input Message*
   - *Field Format* : *Character*
   - *Field Length* : *30*
     Press Enter after every input text.
2. Repeat steps 1 - 4 to add the output field as a child of the *HELLO_SERVER_OV* view. Name this field *OUTPUT_MESSAGE* . This handles the output view. For step 4, enter or change the following fields and press Enter:
   - *Field Format* : *Character*
   - *Field Length* : *256*
3. Because these fields also are included on the end-user window, you must add the fields to the *HELLO_WIN_V* view.
   a. Select both the ***INPUT_MESSAGE*** and ***OUTPUT_MESSAGE*** fields.

      > To select multiple objects, press *Ctrl* while clicking the object names.

   b. Select *Edit* > *Copy* from the Construction Workbench menu. You can also right-click and select *Copy* from the pop-up menu.
   c. Select the *HELLO_WIN_V* view.
   d. Select *Edit* > *Paste* from the Construction Workbench menu. You can also right-click and select *Paste* from the pop-up menu.
      You can also copy fields (and other objects) by pressing *Ctrl* while dragging the fields to the view.
      The system creates a copy of both fields as children of the *HELLO_WIN_V* view.

The hierarchy diagram should look like the one shown in the following figure.

Completed hierarchy diagram



```
Project: HELLO_WORLD
  Function: HELLO_WORLD
    Process: HELLO_CLIENT_PROC
      Rule: HELLO_CLIENT
        Rule: HELLO_SERVER
          View: HELLO_SERVER_IV
            Field: INPUT_MESSAGE
          View: HELLO_SERVER_OV
            Field: OUTPUT_MESSAGE
        Window: HELLO_WIN
          View: HELLO_WIN_V
            Field: INPUT_MESSAGE
            Field: OUTPUT_MESSAGE
```

To save the hierarchy, commit all changes to the repository. Select *File* > *Commit* from the Construction Workbench menu.

You can also commit changes by clicking the *Commit* button in the toolbar or by pressing Ctrl+M on your keyboard.
Now that the application hierarchy has been built, we can create the user interface. Continue with Creating the User Interface.

Creating the User Interface

You use the Window Painter to create the end-user interface for each window in an AppBuilder application. You can create objects within the window (for example, text fields, list boxes, or push buttons) to allow the end-user to view or modify the data and control the application. Window Painter contains extensive tools for customizing the size, color, and other attributes of each object.
The Hello World sample application contains a single window named *HELLO_WIN* . In this section we create the user interface for the application. The tasks in creating the window include:

- Using Window Painter
- Creating a Window
- Adding an Edit Field
- Adding Static Text
- Adding Buttons

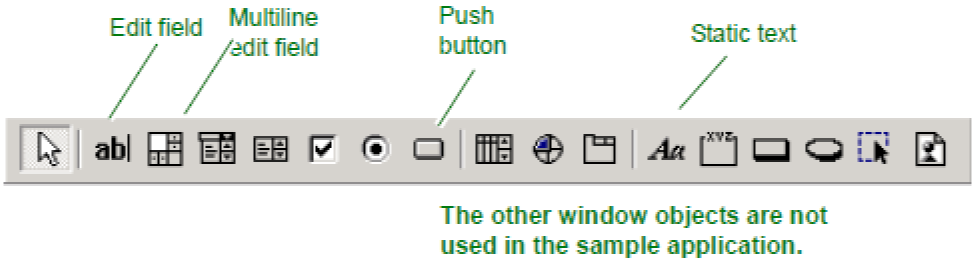
> Depending on your execution environment, you can also use a third-party HTML editor to generate HTML interfaces. The *Development Tools Reference Guide* contains detailed information about creating, importing, and exporting HTML windows.

The *Development Tools Reference Guide* includes a chapter about Window Painter and how to use it to construct a GUI.

Using Window Painter

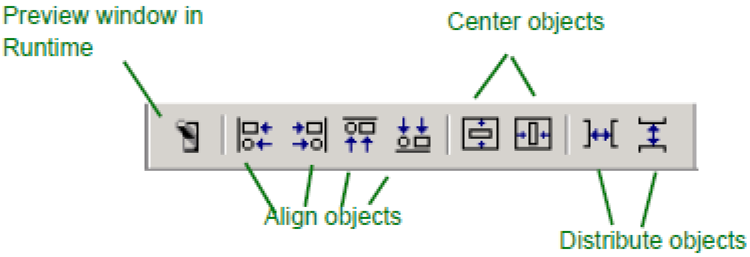
The Window Painter toolbar provides a method for quickly adding interface objects to the window.
To open Window Painter, double-click the window HELLO_WIN in the Hierarchy window.
Select *View* > *Toolbars* > Window - Objects if not already selected from the *Construction Workbench* menu.

Window Objects toolbar


Edit field
Multiline edit field
Push button
Static text
The other window objects are not used in the sample application.

You can use the Window Layout toolbar (Figure 4-2) to quickly and easily align objects in the window. Select one or more objects, then click the appropriate button in the Window Layout toolbar.

Window Layout toolbar



In addition to using the toolbar, you can also manipulate window objects by selecting the object and then selecting *Layout* from the Construction Workbench menu.

Creating a Window

Earlier, you created the window object in the application hierarchy. Use the following procedure to create a graphical form of the *HELLO_WIN* window object:

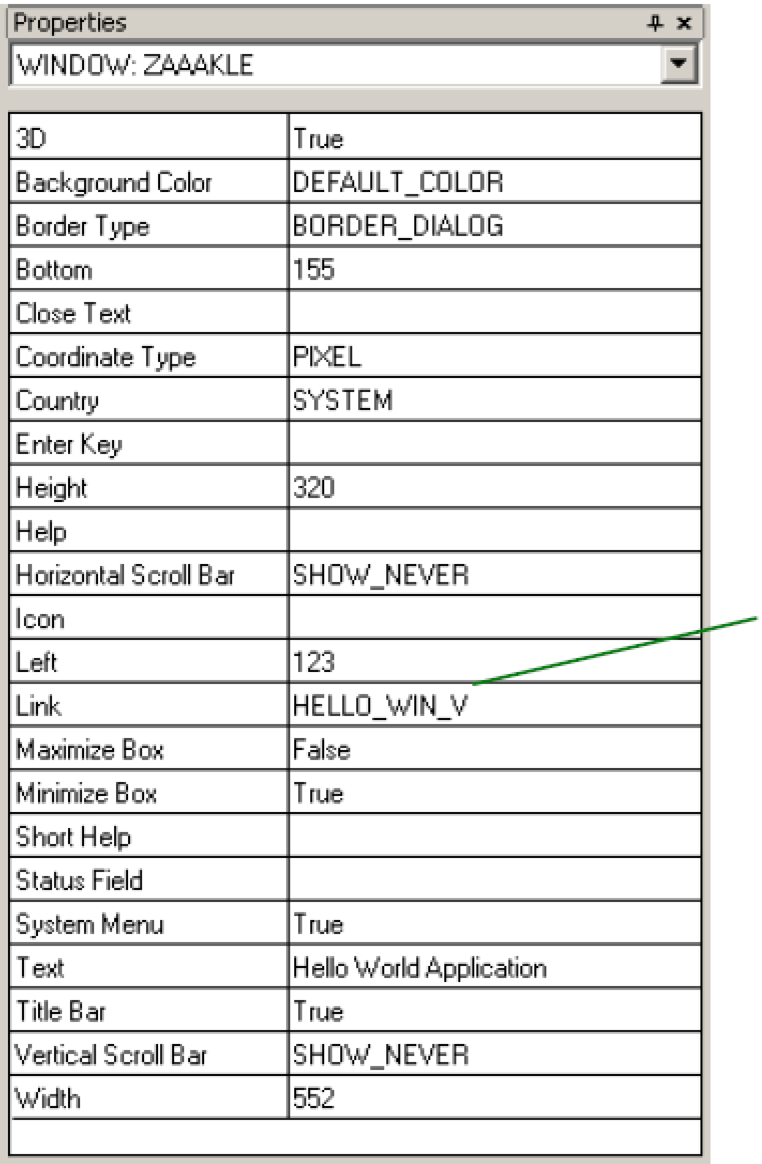
1. Right-click the ***HELLO_WIN*** window on the ***Project*** tab of the Hierarchy and select ***Open Window***.

   > ✓ You can configure the Construction Workbench options to use double-click to show the properties of an object from the Hierarchy window. Select *Tools* > *Workbench Options* from the Construction Workbench menu. Use the *Double-click actions* field on the *Hierarchy* tab to specify the default double-click action.

   The Window Painter displays, showing a blank window. Since the sample application only contains two fields, it might be necessary to resize the window by doing the following:

   a. Click the window.
   b. Click and drag one of the window handles to resize the window, as needed.
2. If the Properties dialog box is not visible (as shown in HELLO_WIN Window properties), right-click the HELLO_WIN blank window from the tool panel and select *Properties* (or double-click the tool window and the Properties box displays). Use the Properties window to control the display attributes for the window.

HELLO_WIN Window properties



Properties

WINDOW: ZAAAKLE

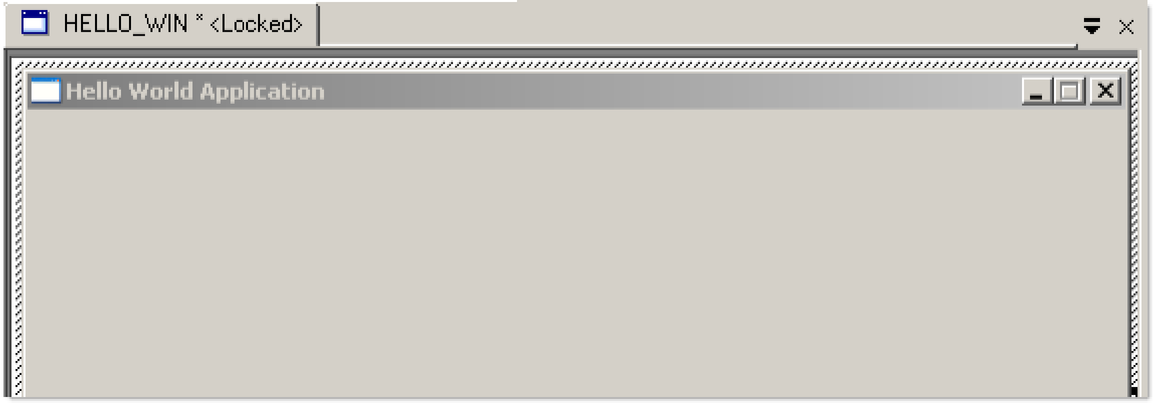
3D	True
Background Color	DEFAULT_COLOR
Border Type	BORDER_DIALOG
Bottom	155
Close Text	
Coordinate Type	PIXEL
Country	SYSTEM
Enter Key	
Height	320
Help	
Horizontal Scroll Bar	SHOW_NEVER
Icon	
Left	123
Link	HELLO_WIN_V
Maximize Box	False
Minimize Box	True
Short Help	
Status Field	
System Menu	True
Text	Hello World Application
Title Bar	True
Vertical Scroll Bar	SHOW_NEVER
Width	552

Use these fields to customize the window—for example, adding colors or scroll bars.

The system automatically links this window to its child view.

3. In the *Text* field, type *Hello World Application* and press *Enter* .
   At runtime, this text displays in the window title bar.
   <Locked> indicates the window is locked in the repository by the current session. * indicates the window has been modified. When you commit your session to the repository, these markers will no longer display.

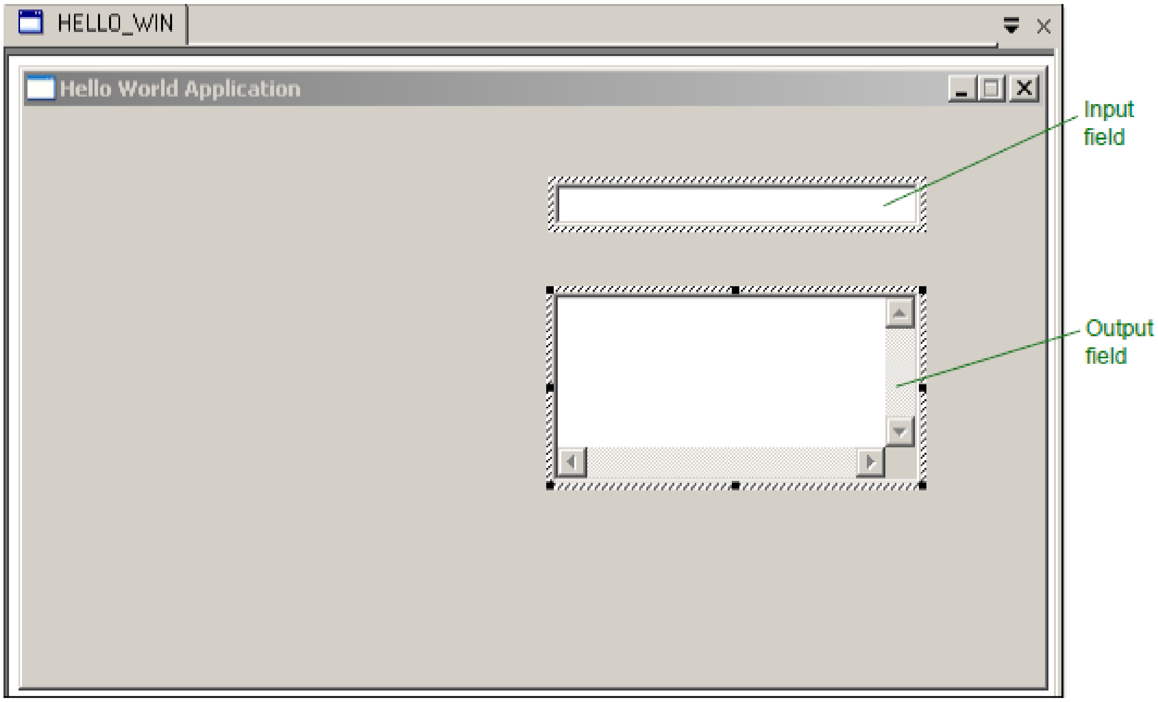
Hello World Application * <Locked> window



Adding an Edit Field

Edit fields create input and output areas in a window. In an AppBuilder application, an edit field can be populated by typing in data at runtime or by mapping data from a rule.

Hello World Application window with fields

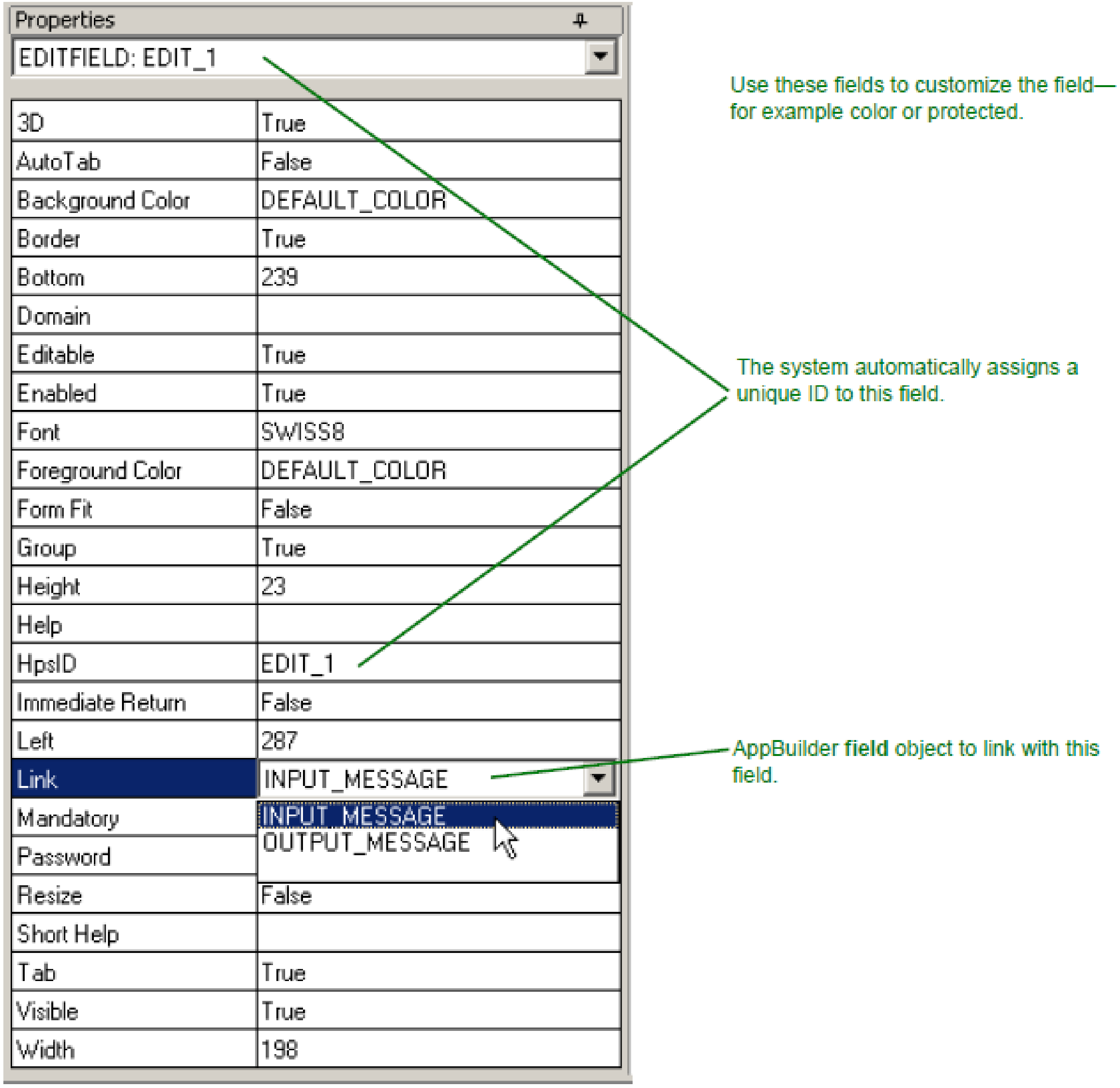


For our sample application, we need to add an input and an output field to the Hello World Application window.

1. Click the ***Edit Field*** button ab| in the window object toolbar and move the cursor to the Window Painter window. The cursor changes to a plus (+).
2. In the Window Painter window, click the *Hello World Application* window to add the field to the window. The system places a blank edit field in the window.
   - To move the field, click and drag the field.
   - To resize the field, click and drag one of the window handles.
3. Select the newly created field. In the Field Properties window select *INPUT_MESSAGE* from the *Link* drop-down list (as shown in Field Properties window) and press *Enter* . This links the display field to the *INPUT_MESSAGE* field in the hierarchy.

> If the Properties window is not visible, right-click the HELLO_WIN window in the Tool window and select *Properties* , or double-click the new field to display the window.

Field Properties window



EDITFIELD: EDIT_1

3D	True
AutoTab	False
Background Color	DEFAULT_COLOR
Border	True
Bottom	239
Domain	
Editable	True
Enabled	True
Font	SWISS8
Foreground Color	DEFAULT_COLOR
Form Fit	False
Group	True
Height	23
Help	
HpsID	EDIT_1
Immediate Return	False
Left	287
Link	INPUT_MESSAGE
Mandatory	INPUT_MESSAGE
Password	OUTPUT_MESSAGE
Resize	False
Short Help	
Tab	True
Visible	True
Width	198

Use these fields to customize the field—for example color or protected.

The system automatically assigns a unique ID to this field.

AppBuilder **field** object to link with this field.

4. Create a mutli-line edit field to handle the application output. Click the *Multiline Edit Field* button in the window object toolbar and move the cursor to the Window Painter window. The cursor changes to a plus (+).
5. Repeat step 2. to add the output field.
   - Link this field to *OUTPUT_MESSAGE* from the Link drop list and press Enter.
   - Change the *Editable* field to *True* .
   - Change the *Word Wrap* field to *True* .
6. To align the fields, select both fields and then select *Layout* > *Align* > *Lef* t from the Construction Workbench menu or click the *Align Left* button in the Window Layout toolbar.

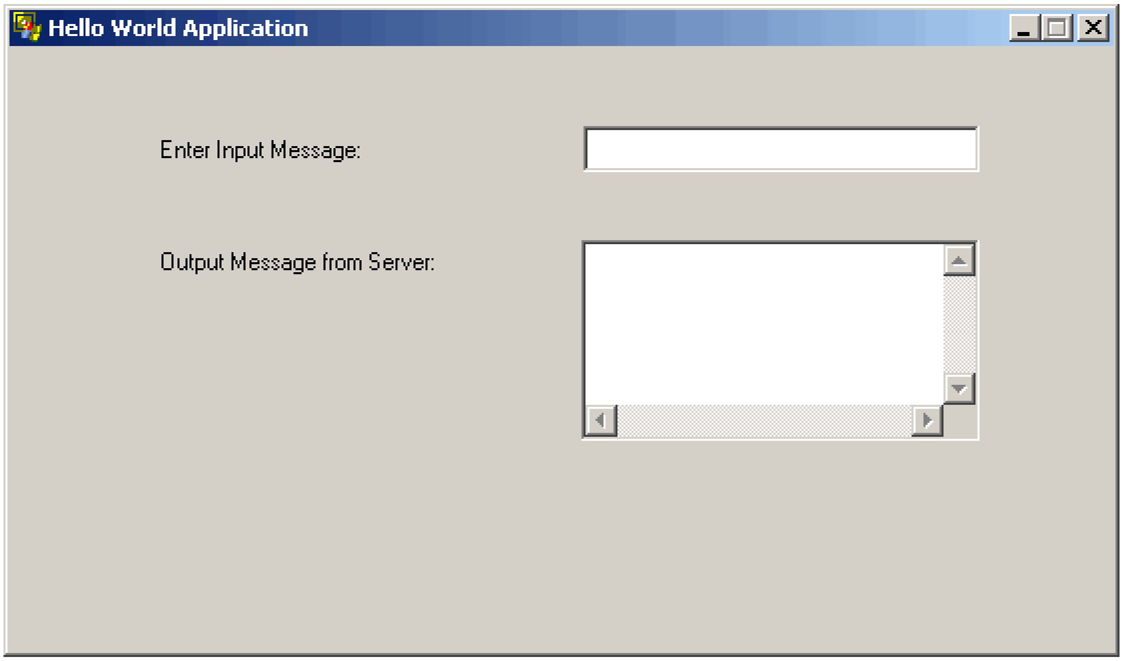
✓ To select multiple fields, press *Ctrl* while clicking the fields.

Your window should look like the one shown in Hello World Application window with fields.
In addition to adding fields by creating them with Window Painter, you can also add fields to a window by dragging them from the hierarchy on the *Project* tab of the Hierarchy window into the Window Painter window.
The system adds the field (and a text label, if the *Screen Literal Long* field of the Properties window of a field is defined as shown in Object Property - General [Field] dialog) to the window. The static text (text label) is not grouped with the field; if you move the field, you must move the text separately.

Adding Static Text

Static text is a text label placed anywhere in a window. These labels are for display only and do not link to a repository field. End users cannot change static text.

Hello World Application window with fields and labels



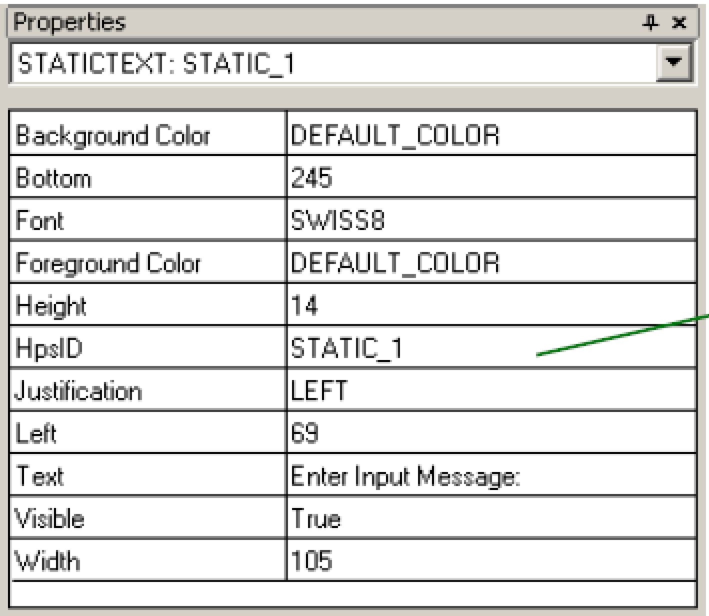
For the sample application, we must add text labels to identify the two fields in the *Hello World Application* window.

> ⓘ If you add fields to the window by dragging them from the hierarchy, the system automatically includes a text label if it is identified in the properties of the field. You can edit or change this label, as needed.

1. Click the ***Static Text*** button Aa in the window object toolbar and move the cursor to the Window Painter window. The cursor changes to a plus (+).
2. In the Window Painter window, click on the *Hello World Application* window object to add the text to the window. The system places a blank text box in the window.
    - To move the text box, click and drag the text box.
    - To resize the text box, click and drag one of the text box handles.
3. In the *Text* field of the Properties window (as shown in Field Property window) remove the word "text" and type *Enter Input Message:* and press *Enter* . This is the text label that displays for the window input field (HELLO_WORLD_INPUT.)

    > ⓘ If the Properties window is not visible, right-click the text box and select *Properties* .

    Field Property window



Properties

STATICTEXT: STATIC_1

Background Color	DEFAULT_COLOR
Bottom	245
Font	SWISS8
Foreground Color	DEFAULT_COLOR
Height	14
HpsID	STATIC_1
Justification	LEFT
Left	69
Text	Enter Input Message:
Visible	True
Width	105

The system automatically assigns a unique ID to this field.

4. Repeat steps 1 - 3, and for step 3, add the following static text for the window output field: *Output Message from Server:*
5. To align the fields, select both fields then select *Layout* > *Align* > *Lef* t from the Construction Workbench menu.

   You can also align objects by selecting the objects then clicking the *Align Left* button in the Window Layout toolbar.

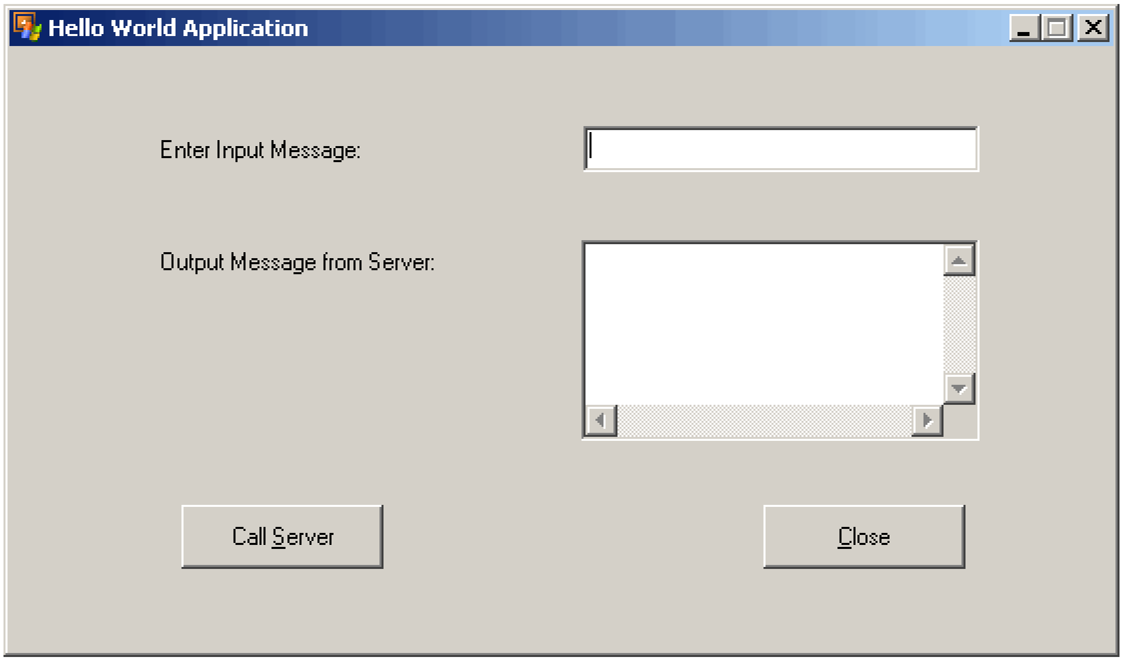
> ✓ To select multiple fields, press *Ctrl* while clicking the fields.

Your window should look like the one shown in Hello World Application window with fields and labels.

Adding Buttons

A push button allows the end-user to trigger an event. End-users can select buttons either by mouse click or an accelerator key. In an AppBuilder application, push buttons are not connected to a specific repository object - so clicking one or using an accelerator key sends an event to a rule.

Completed Hello World Application window



For the sample application, we need two buttons: one to send the end-user input to the server and one to close the Hello World Application window. To create these buttons, complete the following steps:

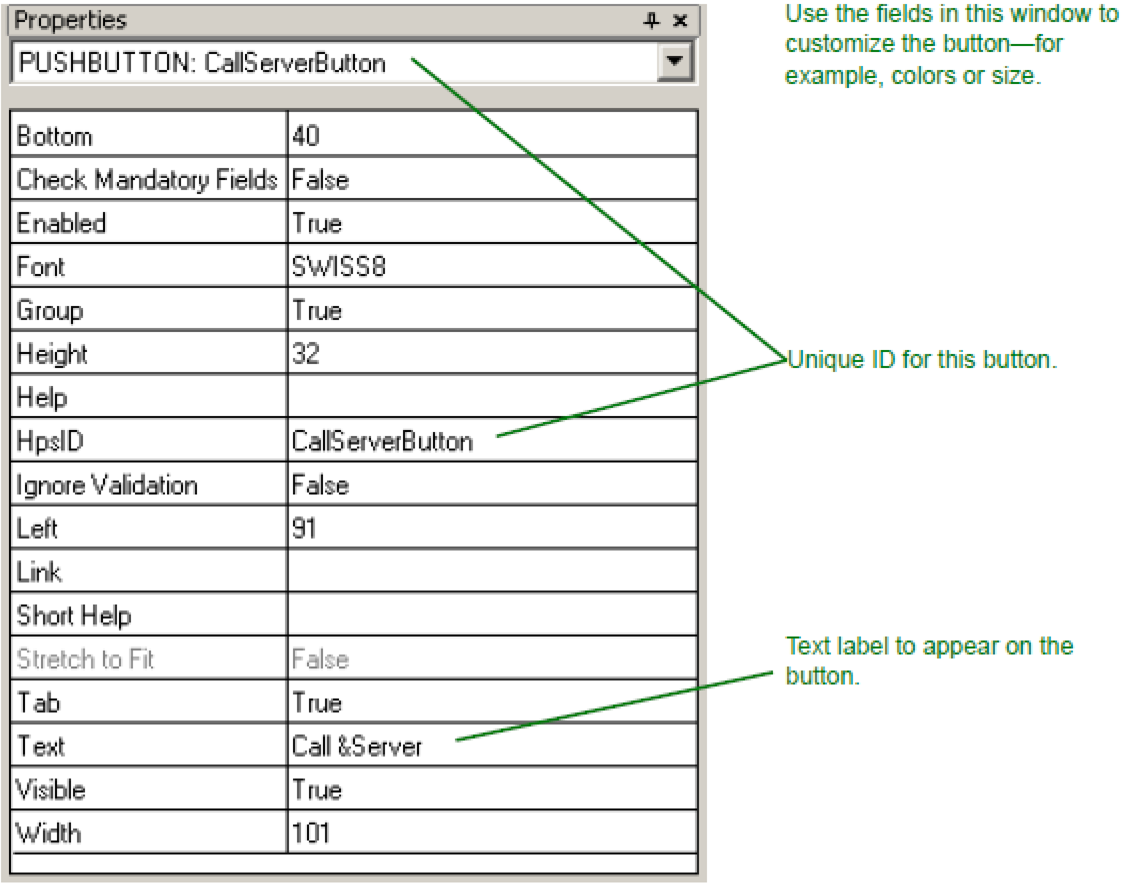
1. Click the ***Push Button*** button in the Window Painter toolbar and move the cursor to the Window Painter window. The cursor changes to a plus (+).
2. In the Window Painter window, click the Hello World Application window to add the button to the window.
   The systems places a blank button named *Push* in the window.
   - To move the button, click and drag the button.
   - To resize the button, click and drag one of the button handles.
3. Use the button Properties window (Button Properties window) to specify the attributes for this button.

Button Properties

Field	What to enter	Description
HpsID	*CallServerButton*	The unique name of this button. AppBuilder uses this ID to reference this button in other views and rules.
Text	*Call &Server*	The text that displays on the button. The ampersand (*&*) identifies the keyboard shortcut for this button (that is, *Alt* + *S* for this button).

If the Properties window is not visible, right-click the button and select *Properties* .

Button Properties window



Properties

PUSHBUTTON: CallServerButton

Bottom	40
Check Mandatory Fields	False
Enabled	True
Font	SWISS8
Group	True
Height	32
Help	
HpsID	CallServerButton
Ignore Validation	False
Left	91
Link	
Short Help	
Stretch to Fit	False
Tab	True
Text	Call &Server
Visible	True
Width	101

Use the fields in this window to customize the button—for example, colors or size.

Unique ID for this button.

Text label to appear on the button.

4. Repeat steps 1 - 3 to add a second button to close the windows. Use the following properties for this button in step 3:
   - *HpsID = CloseButton*
   - *Text = &Close*

Your window should look like the one shown in Completed Hello World Application window.
To preview how the window displays, open the window and select *Layout* > *Preview in runtime* from the Construction Workbench menu or click

the *Preview* button in the Window Layout toolbar.
Be sure that you save changes by Committing Changes.

Committing Changes

To save the window and commit all changes to the repository, do one of the following:

- Select *File* > *Commit* from the Construction Workbench menu.
- Click the *Commit* button

in the toolbar or press Ctrl+M on your keyboard.
Now that the user interface is complete, we must write the rules to control the application. Continue with Creating Rules.

Creating Rules

To create and edit the rules that control how the application executes, use the Rule Painter in the Construction Workbench. Creating a rule is similar to creating a procedure in a computer language, such as C or Cobol. The Rule Painter includes several tools for quickly building and testing rule statements. The tasks in creating rules include the following:

- Using Rule Painter
- Adding a Client Rule
- Adding a Server Rule
- Verifying Rules

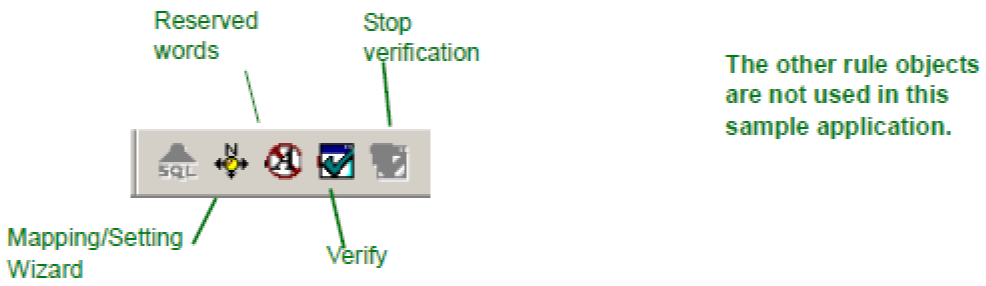
For this sample application, you have already created in the Hierarchy Diagram the rule statements for the two rules, HELLO_CLIENT and HELLO_SERVER, as shown in Completed hierarchy diagram. Now you will create the rules (code).

Using Rule Painter

Rule Painter is basically a text editor that lets you enter, change, delete, and move rule statements within the Rule window. The functionality is similar to context-based source editors.
The Text Editor - Tools toolbar provides a method for quickly adding statements to the rule.

1. To open Rule Painter, double-click HELLO_SERVER in the Hierarchy window.
2. Select *View* > *Toolbars* > Text Editor - Tools if not already selected from the *Construction Workbench* menu.

Text Editor - Tools toolbar


Reserved words
Stop verification
The other rule objects are not used in this sample application.
Mapping/Setting Wizard
Verify

In addition to using the toolbar, you can also use one of the following methods to add items to rules:

- Right-click in the Rule Painter window and select an item from the pop-up menu.
- Click the Rule Painter window and select *Tools* from the Construction Workbench menu.

Adding a Client Rule

For our sample application, the client rule requires two procedures: one to handle the *Call Server* button, and one to handle the *Close* button.

- In this sample application (or Hello World application), when you click *Call Server* , the system calls the *HELLO_SERVER* rule.
- When you click *Close* , the system closes the Hello World Application window.

To add a client rule, follow the steps:

1. Right-click the ***HELLO_CLIENT*** rule on the ***Project*** tab of the Hierarchy window and select ***Open Rule***. Rule Painter displays a blank text window.

   > ✓ You can configure the Construction Workbench options to use double-click to show the properties of an object from the Hierarchy window. Select *Tools* > *Workbench Options* from the Construction Workbench menu. Use the *Double-click actions* field on the *Hierarchy* tab to specify the default double-click action.

2. Add a comment line to identify this rule. Although comments are not required, it is good programming practice.
   a. Type *> and press *Enter* . In the AppBuilder Rules Language, this signifies the beginning of a comment. As you type the comment, the system automatically colors the comments green.
   b. Type *Rule: HELLO_CLIENT* and press *Enter* .
   c. Type <* and press *Enter* to signify the end of comment. The text displays green. Make sure that the > and < are facing the right way.
3. Type the following procedure (Call Server) in the rule window:

   ```
   proc CallServerButtonClick for Click object CallServerButton
   (e object type ClickEvent)
   map INPUT_MESSAGE of HELLO_WIN_V to INPUT_MESSAGE
   of HELLO_SERVER_IV
   use rule HELLO_SERVER
   map OUTPUT_MESSAGE of HELLO_SERVER_OV to OUTPUT_MESSAGE
   of HELLO_WIN_V
   endproc
   ```

   > ✓ As you type the rule, the system automatically colors any reserved words (for example, proc or map) blue.

4. To add the second procedure (Close), type the following procedure in the rule window:

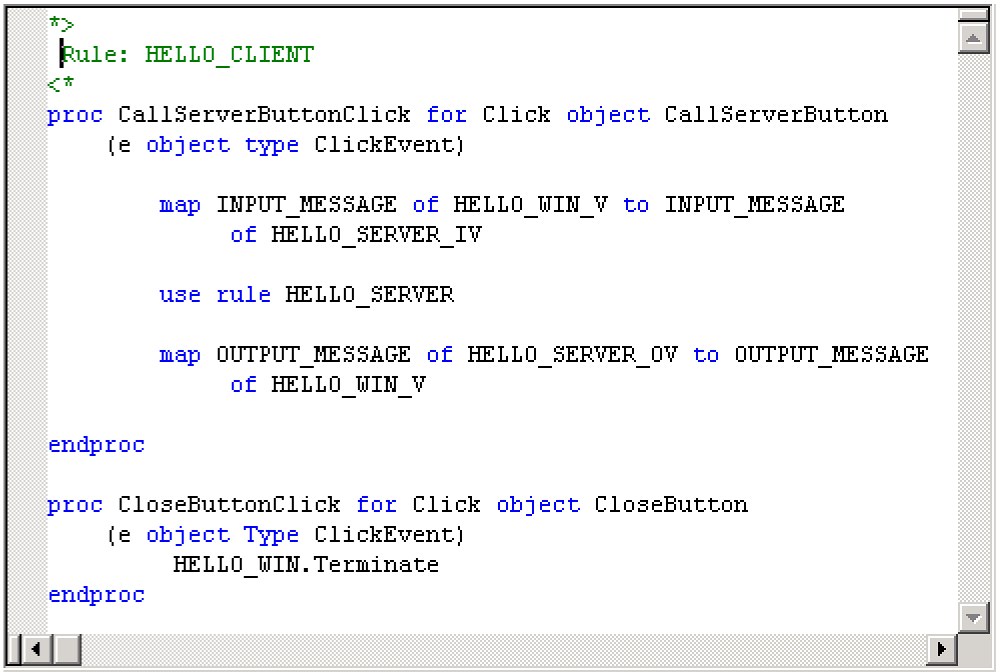
   ```
   proc CloseButtonClick for Click object CloseButton
   (e object type ClickEvent)
   HELLO_WIN.Terminate
   endproc
   ```

   Your window should look like the one shown in HELLO_CLIENT rule.

5. To save the rule, commit all changes to the repository. Select *File* > *Commit* from the Construction Workbench menu.

   You can also commit changes by clicking the *Commit* button 💾 in the toolbar or pressing Ctrl+M on your keyboard.

HELLO_CLIENT rule



```
*>
Rule: HELLO_CLIENT
<*
proc CallServerButtonClick for Click object CallServerButton
    (e object type ClickEvent)

        map INPUT_MESSAGE of HELLO_WIN_V to INPUT_MESSAGE
             of HELLO_SERVER_IV

        use rule HELLO_SERVER

        map OUTPUT_MESSAGE of HELLO_SERVER_OV to OUTPUT_MESSAGE
             of HELLO_WIN_V

endproc

proc CloseButtonClick for Click object CloseButton
    (e object Type ClickEvent)
         HELLO_WIN.Terminate
endproc
```

Adding a Server Rule

For our sample application, the server rule receives the message that the user types and copies it to the OUTPUT_MESSAGE field. This rule also echoes the text in the execution log.

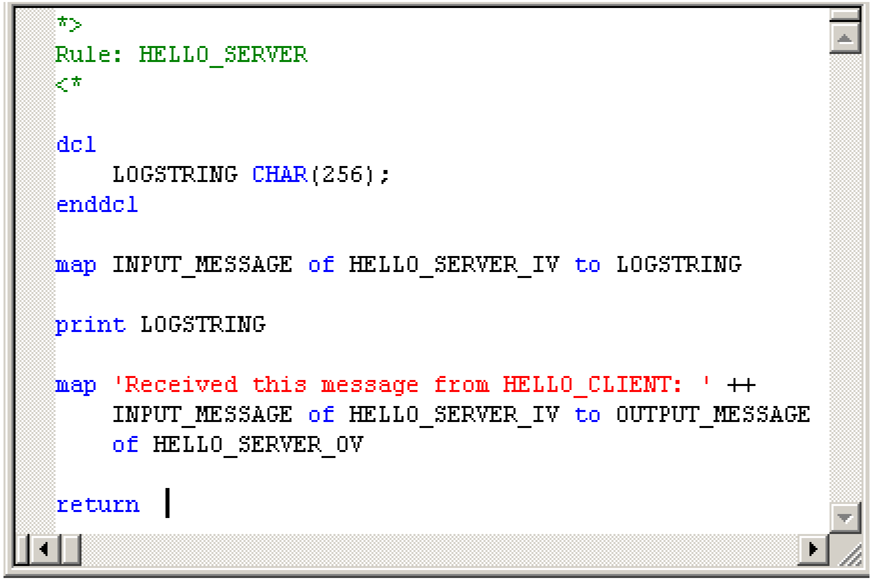
1. Right-click the ***HELLO_SERVER*** rule on the ***Project*** tab of the Hierarchy window and select ***Open Rule***. The Rule Painter displays, showing a blank window.
2. Type the following rule in the rule window:

```
*>
Rule: HELLO_SERVER
<*

dcl
LOGSTRING CHAR(256);
enddcl
map INPUT_MESSAGE of HELLO_SERVER_IV to LOGSTRING
print LOGSTRING
map 'Received this message from HELLO_CLIENT: ' \+\+
INPUT_MESSAGE of HELLO_SERVER_IV to OUTPUT_MESSAGE of
HELLO_SERVER_OV
return
```

Your window should look like the one shown below:

HELLO_SERVER rule



```
*>
Rule: HELLO_SERVER
<*

dcl
    LOGSTRING CHAR(256);
enddcl

map INPUT_MESSAGE of HELLO_SERVER_IV to LOGSTRING

print LOGSTRING

map 'Received this message from HELLO_CLIENT: ' ++
    INPUT_MESSAGE of HELLO_SERVER_IV to OUTPUT_MESSAGE
    of HELLO_SERVER_OV

return
```

To save the rule, commit all changes to the repository. Select *File* > *Commit* from the Construction Workbench menu.

You can also commit changes by clicking the *Commit* button in the toolbar or by pressing Ctrl+M on your keyboard.

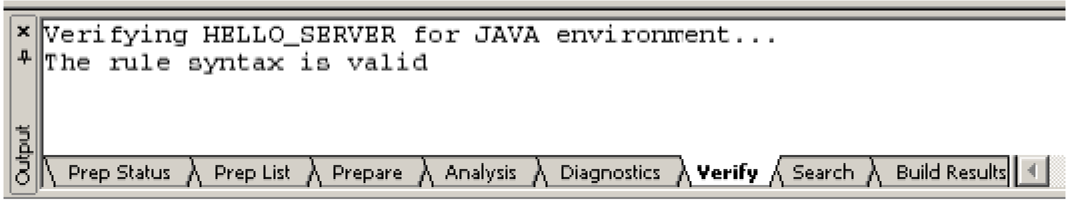
Verifying Rules

Use the *Verification Language* option to specify the execution environment (for example, C, Java, or Cobol) of the rule.

1. Open the Rule Painter window for a rule.
2. Right-click in the Rule Painter window for the rule and select *Verification Language* > *Java* . This ensures that AppBuilder verifies the rule for a Java environment, since you are designing the Hello World Application as a standalone, Java application.
   You can also set the verification language by selecting *Build* > *Verification Language* > *Java* from the Construction Workbench menu.
3. Right-click in the Rule Painter window for the rule and select *Verify* from the pop-up panel.

   You can also verify a rule by selecting *Build* > *Verify* from the Construction Workbench menu or by clicking the *Verify* button in the Text Editor - Tools toolbar.
   The system checks the rule and displays the results on the *Verify* tab of the Status window, as in Verify tab of Status window.

Verify tab of Status window



```
Verifying HELLO_SERVER for JAVA environment...
The rule syntax is valid
```

If there are errors in the rules, double-click the error or warning in the Status window. AppBuilder displays the associated line of the rule in the Rule Painter window.
After verifying that all rules are correct, commit all changes to the repository. Continue with Preparing and Executing the Application.

Preparing and Executing the Application

When AppBuilder prepares your application, it performs the following functions:

- Transforms the rules you have written in the Construction Workbench into an executable program
- Compiles the source code for any third-generation language components used by your rules
- Prepares any files your rules or components access, and creates their corresponding database tables
- Prepares the sets the application uses
- Makes available to the runtime environment the menus, icons, windows, and workstation reports that comprise your application end-user interface

Tasks in preparing and executing the application include the following:

- Setting Preparation Options
- Preparing the Application
- Executing the Application

Depending on your planned deployment and execution environment, you may also be able to prepare mainframe rules for test execution on the workstation or check the syntax of mainframe rules.
You can prepare a single element in the hierarchy (for example, a rule or function) or the complete application.

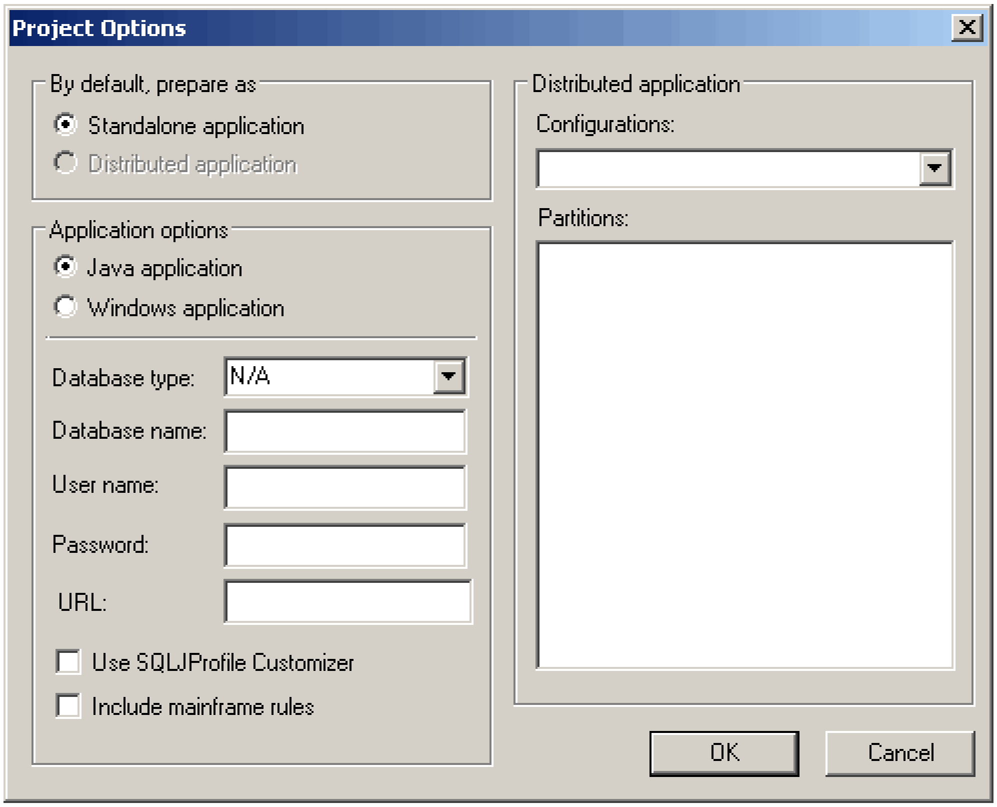
Setting Preparation Options

We prepare the sample application as a standalone Java application.

1. Select ***File > Project Options*** from the Construction Workbench menu.
   The Project Options dialog displays, as shown in the following figure.
   You can also access the Project Options dialog by right-clicking an object on the *Project* tab and selecting *Project Options* from the pop-up menu or by clicking the *Project Options* button in the toolbar.

   > By default, the system uses the objects specified when you created the project. See Creating the Project and Create New Project window.

   Project Options dialog

   Project Options

   By default, prepare as
   ● Standalone application
   ○ Distributed application

   Application options
   ● Java application
   ○ Windows application

   Database type: N/A
   Database name:
   User name:
   Password:
   URL:
   □ Use SQLJProfile Customizer
   □ Include mainframe rules

   Distributed application
   Configurations:
   Partitions:

   * *

2. Configure the following options for the Hello World Application.

Field	What to enter	Description
By default prepare as	*Standalone application*	The sample application runs locally on your PC and does not require connection to a database or server.
Application options	*Java application*	AppBuilder prepares the application and rules to execute in a Java environment.

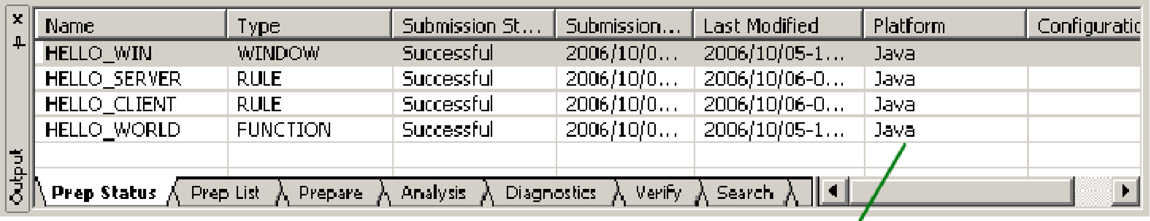
Database type	N/A	The sample application runs locally on your PC and does not require connection to a database or server. Leave the other database fields (*Database name* , *User name* , *Password* and URL) blank.
Use SQLJProfile Customizer	N/A	This utility augments the profile with DB2-specific information for use at run time.The SQLJ translator performs analysis on the SQLJ source file by checking for incorrect SQLJ syntax. For the sample application, this box does not need to be checked.
Include mainframe rules	N/A	The sample application runs locally on your PC and does not require connection to a mainframe host.
Configurations	N/A	Since you are using a standalone application, there is no need to set configurations.
Partitions	N/A	Since you are using a standalone application, there is no need to set partitions.

3. Click *OK* .

Preparing the Application

The Construction Workbench contains two commands for preparing applications: *Prepare* and *Super Prepare* . The Prepare function prepares only the selected object. The Super Prepare function prepares the selected object and its children.
For the sample application, we Super Prepare the *HELLO_WORLD* function, thereby preparing the entire project.
To Super Prepare the project, select the *HELLO_WORLD* function in the *Project* tab of the Hierarchy window and select *Build* > *Super Prepare* from the Construction Workbench menu. The system displays the status of each object in the *PrepStatus* tab of the Status window, as shown in the following figure.
You can also prepare the function by right-clicking the *HELLO_WORLD* function and selecting *Super Prepare* from the pop-up menu.

Status window with Prep Status tab



Name	Type	Submission St...	Submission...	Last Modified	Platform	Configuratic
HELLO_WIN	WINDOW	Successful	2006/10/0...	2006/10/05-1...	Java	
HELLO_SERVER	RULE	Successful	2006/10/0...	2006/10/06-0...	Java	
HELLO_CLIENT	RULE	Successful	2006/10/0...	2006/10/06-0...	Java	
HELLO_WORLD	FUNCTION	Successful	2006/10/0...	2006/10/05-1...	Java	

Identifies the currently selected configuration mode.

If any object does not prepare successfully, right-click the object and select *Details* . The system displays a detailed error report for the object.

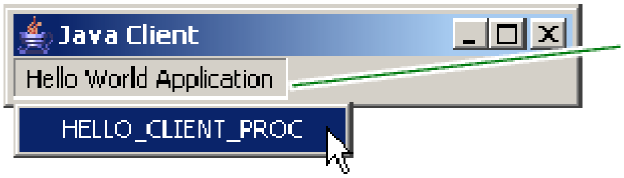
The *Prep List* tab maintains a history of each *Prepare* and *Super Prepare* command submitted.

Executing the Application

After preparing the application successfully, commit all changes to the repository. Select *File* > *Commit* from the Construction Workbench menu.

1. Select ***Run > Java*** from the ***Construction Workbench*** menu.
   The Java Client displays, as shown in the following figure:

   Java Client

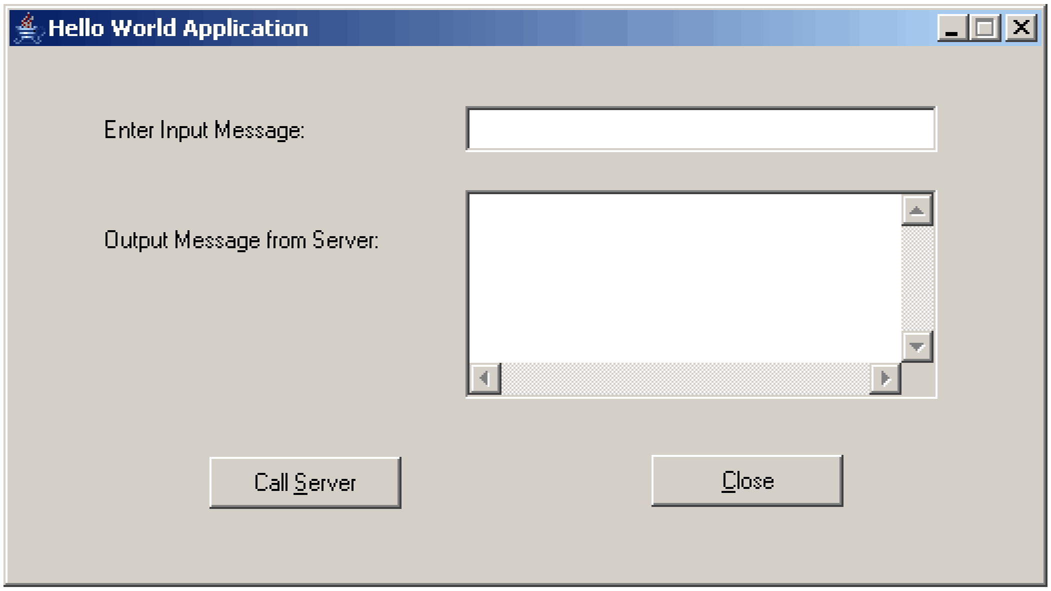


Menu bar description from the HELLO_WORLD function

2. Click *Hello World Application*. A drop down list opens. Click *HELLO_CLIENT_PROC*.
   The system displays the following message: Do you wish to start RuleView?
   RuleView is a debugger that can be used to test and debug an AppBuilder rule. Refer to the *Debugging Applications Guide* for details on using RuleViewer.

3. Click *No* .
   The Hello World Application window displays, as shown in the following figure:

Hello World Application



* *

4. Type your message in the *Enter Input Message* field and click *Call Server* .
   The system copies your message to the *Output Message from Server* field.
5. To close the application, click *Close* .

Congratulations! You have successfully developed an application using the AppBuilder environment. Refer to the other guides in the AppBuilder library for additional information on developing custom applications for your business.

Creating the Configuration Hierarchy

For the sample application, we created a Java application that ran locally on your PC. This standalone application did not require a deployment configuration. However, AppBuilder projects can be built for a number of different configurations. This section explores some of the possible configurations if you were to deploy the sample application to other runtime environments (that is, a distributed application).
Although you can build the configurations in this section, the application will not run from your PC. These procedures are only examples of the preliminary steps involved in developing an application. Deploying them on other machines might involve further configuration. For information about deployment of an application, refer to the *Deploying Applications Guide* .
AppBuilder installs an example file, SAMPLES/HELLO_WORLD.ZIP. The files in this example may be useful to view for comparison. The configurations discussed in this chapter are based on using those sample files.

> The sample files provided in the zip file must be imported into a repository, using the Repository Administration Tool. For further information and details on the import process, refer to *Repository Administration Guide for Workgroup and Personal Repositories* .

The configuration hierarchy associates client processes and logical servers with databases and physical machines, creating an Application Configuration Object. An application configuration groups a number of *partitions* , typically belonging to a single deployment configuration for an AppBuilder project. These partitions contain the necessary information to prepare the objects of the application hierarchy to the correct locations in a preparation network. Use the *Configuration* tab of the Hierarchy window to build the configuration hierarchy.
Tasks in creating the configuration hierarchy include the following:

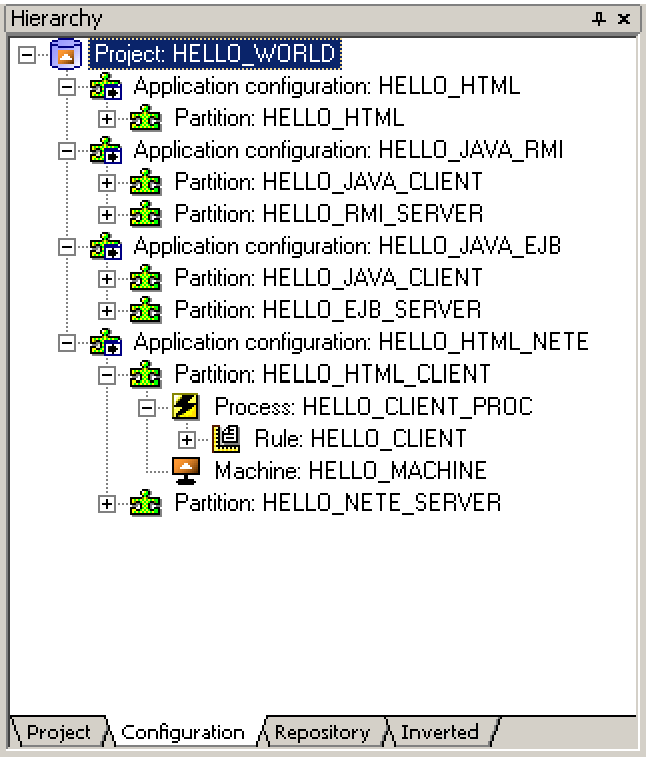
- Using the Configuration Tab
- Creating Sample Configurations

For detailed information about configuration options, refer to the *Deploying Applications Guide* . See also Application Configuration Object and Partition.

Using the Configuration Tab

When a project has been opened, a Project and a Configuration tab display in the Hierarchy window. Click the Configuration tab of the Hierarchy window to display the configuration hierarchy. Similar to the Project tab, the Configuration tab shows a hierarchy of information about the current project. The configuration objects can be expanded to show the objects that make up that configuration. The example in the following figure shows a sample configuration hierarchy with multiple partitions.

Configuration diagram


Hierarchy
Project: HELLO_WORLD
Application configuration: HELLO_HTML
Partition: HELLO_HTML
Application configuration: HELLO_JAVA_RMI
Partition: HELLO_JAVA_CLIENT
Partition: HELLO_RMI_SERVER
Application configuration: HELLO_JAVA_EJB
Partition: HELLO_JAVA_CLIENT
Partition: HELLO_EJB_SERVER
Application configuration: HELLO_HTML_NETE
Partition: HELLO_HTML_CLIENT
Process: HELLO_CLIENT_PROC
Rule: HELLO_CLIENT
Machine: HELLO_MACHINE
Partition: HELLO_NETE_SERVER
Project
Configuration
Repository
Inverted

See the following topics for more information:

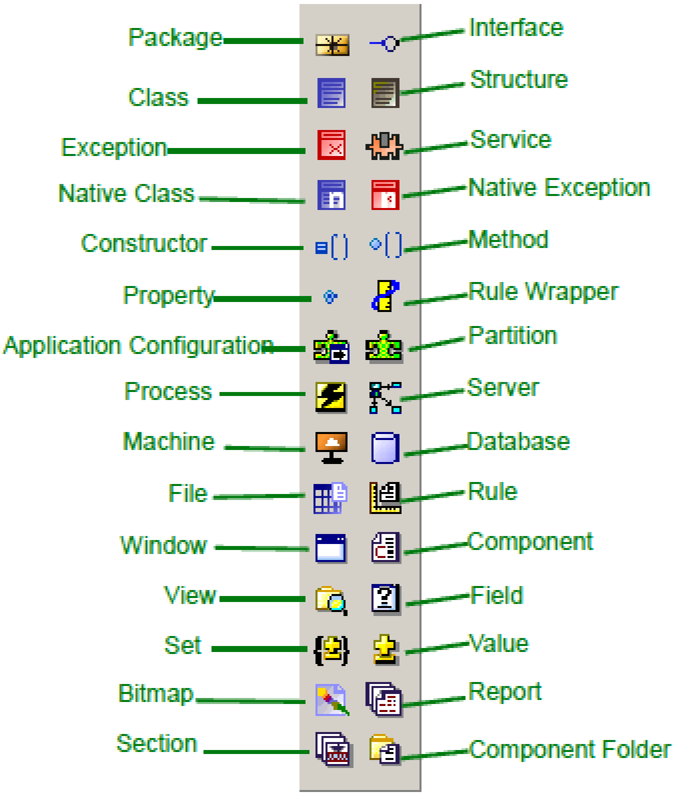
- Hierarchy - Objects Toolbar
- Application Configuration Object
- Partition

Hierarchy - Objects Toolbar

The Hierarchy - Objects toolbar provides a method for quickly adding configuration objects to your configuration hierarchy.

1. To display the toolbar, select ***View > Toolbars*** > ***Hierarchy - Objects*** from the ***Construction Workbench*** menu (see Hierarchy Objects toolbar (used with Configuration tab)).

Hierarchy Objects toolbar (used with Configuration tab)



You can also add objects to the configuration hierarchy by using one of the following methods:

- Right-clicking an object in the configuration diagram and selecting *Insert* .
- Selecting an object in the configuration diagram and selecting *Insert* from the Construction Workbench.

For detailed information about configuration objects, refer to the *Deploying Applications Guide* .

Application Configuration Object

An application configuration object groups a number of partitions units, typically belonging to a single deployment configuration. This object contains the information needed to prepare a distributed application, to migrate it to a production environment, and to administer the application at runtime. Typically, each project contains a single application configuration. For detailed information on configuration objects, refer to the *Deploying Applications Guide* .

Partition

Partitions define the associations between a *client* or *server* and its associated machines or databases. Each partition must be associated with an application configuration.

- Client partitions ? Client partitions contain the processes of a project that execute on the client-side. When you add a process from the project hierarchy to the configuration, AppBuilder automatically includes all the necessary child objects (for example, rules or views). In addition, the client partition includes a machine object that indicates the client runtime environment.
- Server partitions ? Server partitions include a *server* object that defines the type of server (for example, an EJB server). The server partition also includes a machine object and the processes and rules that execute on the server-side.

During preparation, use the Project Options dialog (as shown in Project Options dialog) to select the specific configuration or partitions to use.

Creating Sample Configurations

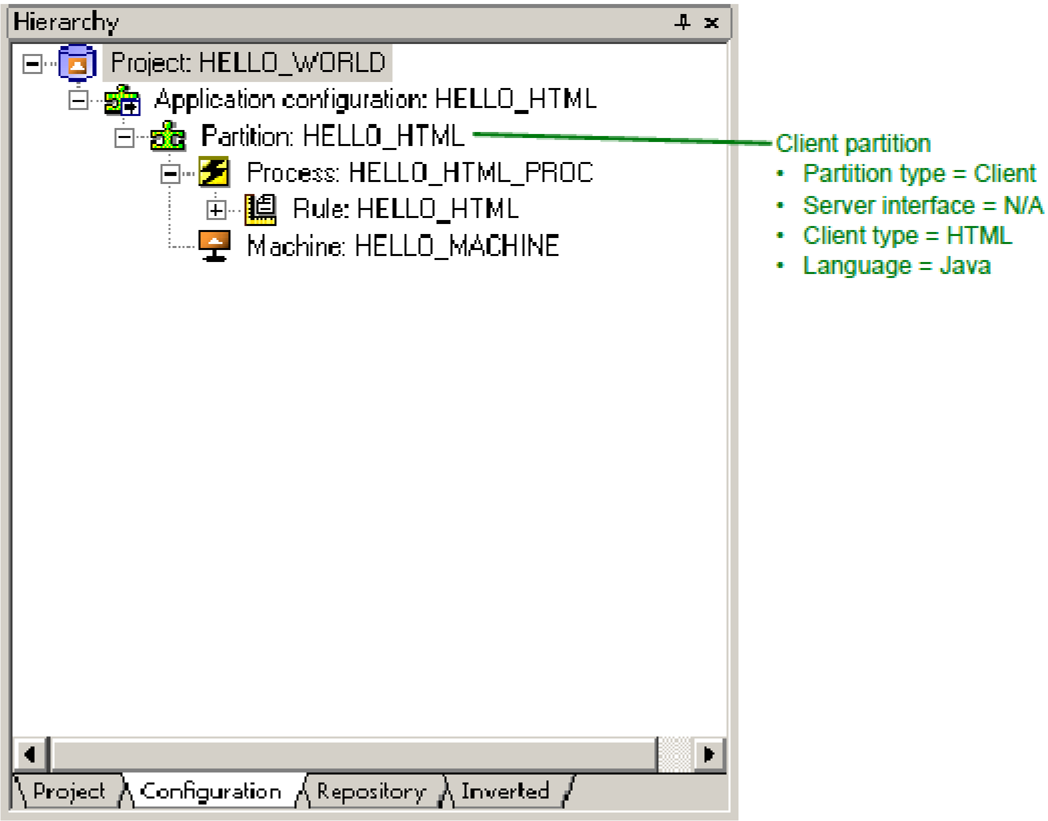
The standard AppBuilder installation includes the following sample configurations for the sample *Hello World Application* . Refer to the *Deploying Applications Guide* for information on using these configurations.

- Thin Client
- EJB Server (and Java Thick Client)
- Servlet (C Server with HTML Client)
- RMI Server with Java Client

Thin Client

When creating an application configuration for a thin client (HTML) application, the client configuration requires a machine object (except when the client is prepared locally). The client partition should also contain the process object that contains the application rules. The following figure illustrates a thin client partition:

Sample thin (HTML) client configuration



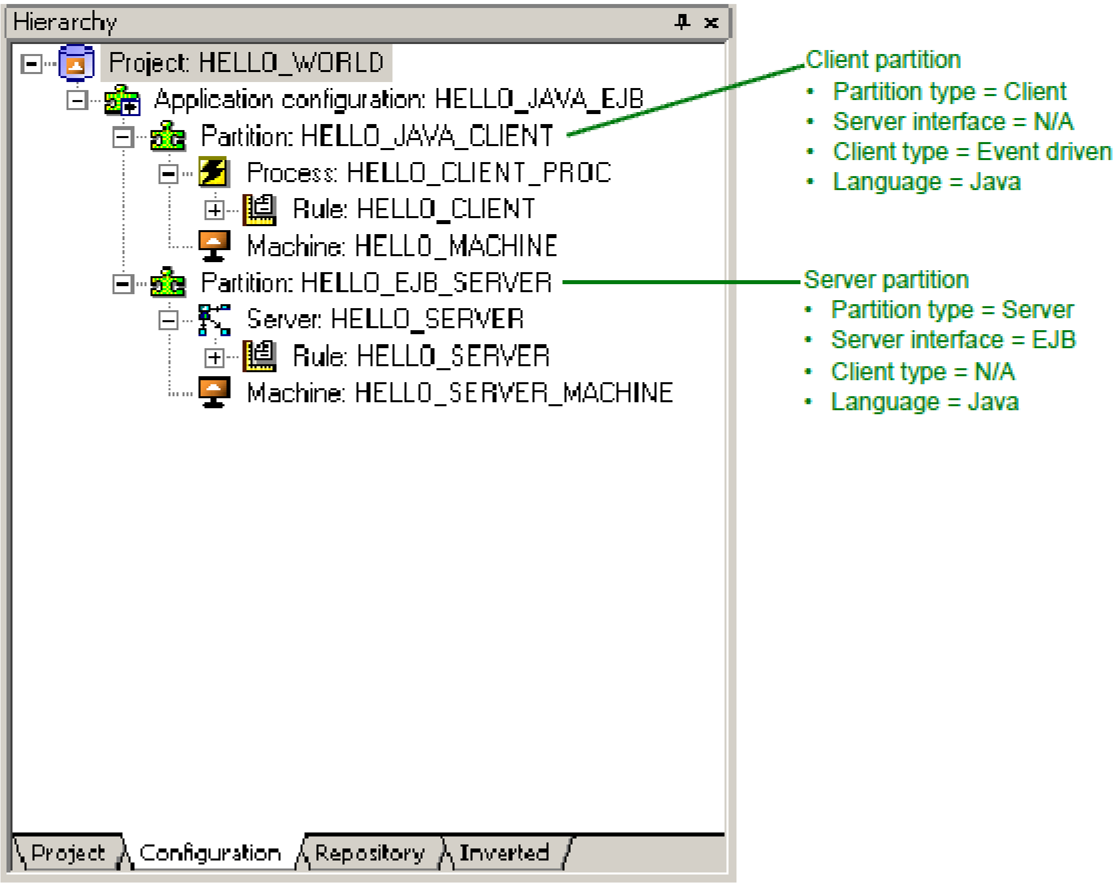
Client partition

- Partition type = Client
- Server interface = N/A
- Client type = HTML
- Language = Java

EJB Server (and Java Thick Client)

When preparing an AppBuilder application to be deployed to a Java Enterprise Java Bean (EJB) server, configure the server interface for the server partition as *EJB* . Sample EJB server (with Java client) configuration illustrates a sample Java client and EJB server configuration.

Sample EJB server (with Java client) configuration



- Project: HELLO_WORLD
  - Application configuration: HELLO_JAVA_EJB
    - Partition: HELLO_JAVA_CLIENT
      - Process: HELLO_CLIENT_PROC
        - Rule: HELLO_CLIENT
      - Machine: HELLO_MACHINE
    - Partition: HELLO_EJB_SERVER
      - Server: HELLO_SERVER
        - Rule: HELLO_SERVER
      - Machine: HELLO_SERVER_MACHINE

Client partition

- Partition type = Client
- Server interface = N/A
- Client type = Event driven
- Language = Java

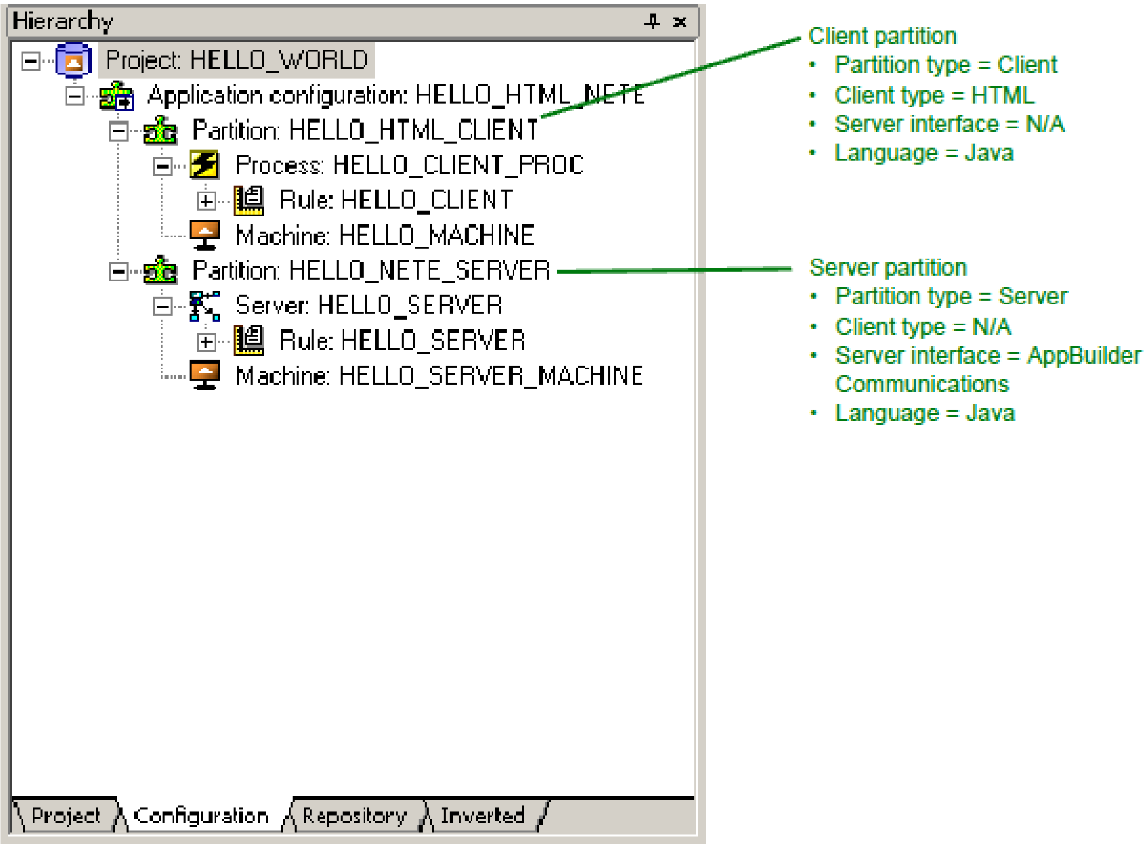
Server partition

- Partition type = Server
- Server interface = EJB
- Client type = N/A
- Language = Java

Servlet (C Server with HTML Client)

When preparing a servlet-based application AppBuilder creates the necessary Java classes, HTML, CSS, JavaScript, and images for the client. The following figure illustrates a sample HTML client and C server configuration.

Sample Servlet configuration



```
Project: HELLO_WORLD
    Application configuration: HELLO_HTML_NETE
        Partition: HELLO_HTML_CLIENT
            Process: HELLO_CLIENT_PROC
                Rule: HELLO_CLIENT
            Machine: HELLO_MACHINE
        Partition: HELLO_NETE_SERVER
            Server: HELLO_SERVER
                Rule: HELLO_SERVER
            Machine: HELLO_SERVER_MACHINE
```

Client partition

- Partition type = Client
- Client type = HTML
- Server interface = N/A
- Language = Java

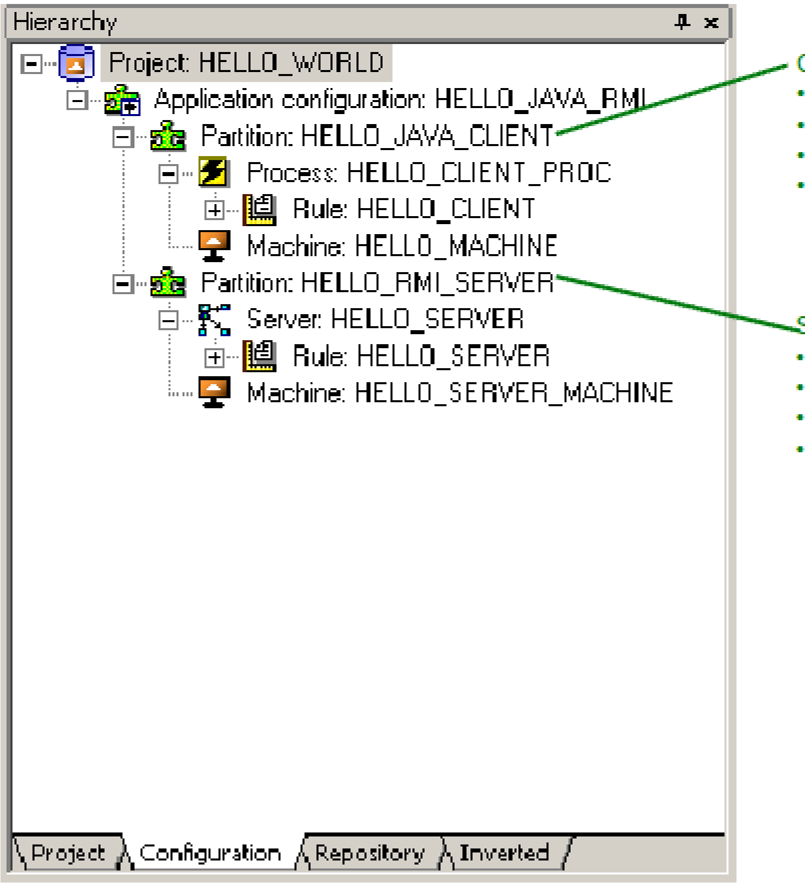
Server partition

- Partition type = Server
- Client type = N/A
- Server interface = AppBuilder Communications
- Language = Java

RMI Server with Java Client

The following figure illustrates a sample RMI server and Java thick client configuration.

Sample RMI configuration



Project: HELLO_WORLD
Application configuration: HELLO_JAVA_RMI
Partition: HELLO_JAVA_CLIENT
Process: HELLO_CLIENT_PROC
Rule: HELLO_CLIENT
Machine: HELLO_MACHINE
Partition: HELLO_RMI_SERVER
Server: HELLO_SERVER
Rule: HELLO_SERVER
Machine: HELLO_SERVER_MACHINE

Client partition

- Partition type = Client
- Client type = Event driven
- Server interface = N/A
- Language = Java

Server partition

- Partition type = Server
- Client type =
- Server interface = RMI
- Language = Java

Product Glossary

Product Glossary

This glossary defines terms used in the AppBuilder environment.

The terms are listed alphabetically.

Numbers	G-H	R
A	I-J	S
B	K-L	T
C	M	U-V
D	N	W-X-Y-Z
E	O	
F	P-Q	

Numbers

3270 Converse

An AppBuilder runtime product that provides a user interface on 3270 terminals - also called non-programmable terminals (NPTs) - which are character-based (in contrast to pixel-based workstations). 3270 Converse uses the same data definitions as on the workstation display panels, although 3270 terminals cannot use some of the more advanced workstation features. The AppBuilder environment supports systems that run on both workstations and 3270 terminals.

A

agent

Any process that performs some communications service, such as eventing. See also *AppBuilder Communications* .

alternate identifier

A non-primary identifier. Part of an entity. See also *primary key* .

API

See *application programming interface (API)* .

AppBuilder Communications

The internal communications interface for the AppBuilder environment.

AppBuilder environment

The collection of systems and tools that comprises the system development environment within which analysts and programmers create software and the execution environment where end users access completed applications.

AppBuilder program group

A program group on your desktop that contains icons that you click to launch the *Construction Workbench* , *repository administration* applications, or other AppBuilder programs.

AppBuilder system service

Single process that handles AppBuilder communications. See Management Console.

application configuration

A configuration object that groups a number of *partition* s, typically belonging to a single deployment configuration for an AppBuilder project.

application programming interface (API)

An interface that allows an application program to use specific data or functions of the interfacing program.

associative entity

A type of entity used in an *Entity Relationship Diagram (ERD)* between two or more fundamental (kernel) entities. It stores information about a *relationship* .

attribute

The smallest unit of information that describes a single characteristic (such as name, kind, usage, or other details) of an entity object in an *Entity Relationship Diagram (ERD)* . An attribute in an entity/attribute hierarchy is shown as a child of the entity it is providing information for. Attributes can also be displayed within entity rectangles in the *Entity Relationship Diagram tool* . See also *attribute entity* and property.

attribute entity

An entity type in the *Information Model* that is a single characteristic of any of the following entity types: *entity* , *relationship* , *identifier* , *attribute* , or *data type* .
For example, a *CUSTOMER* entity might be defined by attributes called *FIRST_NAME* , *LAST_NAME* , *STREET_ADDRESS* , and so forth.
See also *data type* , *foreign key* , and *primary key* .

audit information

Information stored in a *repository* containing the user name and the name of the project that owns an object, along with the date and time it was created and last maintained.

autostart server

A server technology that uses a third-party *daemon* - INETD for TCP/IP, SNA Server/6000 for LU6.2 - to spawn an instance of the server for each connection request it receives; the server instance ends when the connection is closed. Until the connection with the client is closed, the server instance is dedicated to satisfying service requests from that client. See also *banking server* and *forking server* .

B

banking server

A server technology that uses a pre-spawned worker for each connection request it receives; the worker persists after the connection is closed. Additional workers are spawned as necessary. Until the connection with the client is closed, the worker is dedicated to satisfying service requests from that client. See also *autostart server* and *forking server* .

base objects

For MLUI development, the base objects (windows and sets) are the initial source of the newly-created language panels and display values. The base window or set objects refer to one of the following:

- the first window or set created in AppBuilder MLUI
- for windows and sets created in a pre-MLUI AppBuilder, the existing windows and sets are marked as the base objects when a second language is added to that base window or set

breakpoints

A point you set in AppBuilder's debugging tool, *RuleView* , to halt your application so you can examine, and modify, the data in any of its views.

business object

An entity type in the *Information Model* that is a data entity, or set of related data entities, linked to manual and automated processes that create and manipulate the data. Events initiate the processes tied to the data entities. Thus, a business object consists of the following:

- A data object shown in a data object *Entity Relationship Diagram (ERD)*
- All of its state objects shown in *State Transition Diagram*
- All of the process objects for the states shown in a set of *Process Dependency Diagram (PDD)*

C

candidate identifier

An *identifier* designated as a potential *primary key* . The elements of a candidate identifier must uniquely identify the entity.

cardinality

A method for defining the kind of numeric relationship between two entities. The following table identifies the types of cardinality.

Cardinality	Description and example
One to one (1:1)	A customer HAS one account.
Zero or one (0:1)	Also called optional-one. A business CAN BE OWNED BY a parent company.
Zero of many (0:M)	Also called optional-many. A parent company CAN OWN many companies
One to many (1:M)	A customer PLACES many trades.
Many to many (M:M)	Customers OWN many securities.

characteristic entity

A dependent type of *entity* that requires an instance of another entity in the model.
For example, *Customer Address* requires an instance of the *Customer* entity.
A characteristic entity type contains a *kernel entity* type's repeating or optional attributes. Because a characteristic entity type must include its base kernel entity's identifier in its own identifier, characteristic entity types are always dependent. See also *attribute* and *identifier* .

chart object

A *Window Painter* object that graphically displays occurring views.

check box

A *Window Painter* object that allows end users to indicate an on/off or yes/no condition. A check box is a small square that contains a check mark when users select the *On* or *Yes* condition. You can use check boxes singly or in groups to indicate choices that users can select in any combination. You can also use the information users provide in check boxes to set flags.

CICS

Customer Information Control System - a family of application servers and connectors for online transaction management and connectivity from IBM.

class

For object-oriented programming, a class is defined as a template for building objects with identical properties (variables) and methods (functions).

clear

To remove an item from the display but not *delete* it from the *repository* .

client

A system that depends on a server to provide it with programs or access to programs.

code page

A form of character encoding that defines the characters your workstation displays. Each code page assigns a number (1 to 255) to every letter, number, symbol, and other character common to a particular country.

code reuse analysis (CRA)

A method of measuring the number of objects in an application that are used more than once rather than being recreated. See also *reuse* .

code, reentrant

See *reentrant*.

collapse

An action in the *Hierarchy window* that "hides" objects below the object you select to collapse. Use the *explode* action to restore a collapsed object.

column

An object type in the *Information Model* that represents a column in a relational table. One or more attributes translate to a column that all instances of the object type share. See also *attribute* , *relational model* , and *table* .

combo box

A *Window Painter* object that combines the functions of an *edit field* and a drop-down *list box* , which displays a list of objects or settings choices that users can scroll through and select from to fill in the edit field. Or, users can type their choice in the edit field, which need not match the choices in the list if the combo box is not "protected." The combo box is linked to a *field* , which is linked in turn to a *set* that contains the displayed values.

commit

An action of saving to the *repository* any changes you make in a workbench session since startup or the last commit or *rollback* .

component

An object type in the *Information Model* that contains code written in a third-generation computer language (such as C, COBOL, JAVA, or PL/I) to do things the *Rules Language* either cannot do or cannot do quickly enough. This might be a complicated arithmetic algorithm, such as exponentiation, non-SQL data access logic, such as an IMS database interface, or hardware-specific functions, such as time/date stamping. Components also support system re-engineering, whereby existing applications can be defined to the Enterprise Repository. Because components are written in a language specific to one environment, they are not portable between environments. AppBuilder provides some system components as an extension of the *Rules Language* .

Component Folder

A container for external files that support components.

Configuration Tab

A *Construction Workbench* tool in the Hierarchy window for mapping logical servers and client processes to databases and physical machines to specify how an AppBuilder application is to be deployed on target machines. It encapsulates the information you need to prepare the application, distribute it, and administer it at runtime.
See also *logical server* .

connector

An *object* in a *Process Dependency Diagram (PDD)* that generically describes a line that connects two objects together.

construction tools

In AppBuilder in the Construction Workbench, these are the tools to construct an application from what has already been designed using drawing tools. These tools include the following:

- Report Painter
- Rule Painter
- Window Painter

Construction Workbench

The AppBuilder development environment that you can use to design the application, define the hierarchy, create the end-user interface, and construct deployment configurations. You can also use the Construction Workbench to *prepare* objects on local or remote machines. The Construction Workbench includes drawing tools, construction tools, and the RuleView debugger.

Constructor

An operation inside the class that creates the object and/or initialize its state.

converse

A verb in the *Rules Language* and a relationship in the *Information Model* that defines what a *rule* does to display a window or print a report.

copybook

A term originating in the COBOL environment for a file included in the source code of a component or subroutine. In AppBuilder applications, a copybook typically contains a definition of the data structures that the component or subroutine use.

CRUD matrix

A matrix that maps one kind of *entity* against another to show the *relationship* between an entity and the process that creates, reads, updates, and deletes it.

- *C* (Create) - The entity is created with a new instance.
- *R* (Read) - The entity is read to obtain data, check validity, or to get a related entity.
- *U* (Update) - The entity data content or relationships are changed.
- *D* (Delete) - The entity instance is deleted.

current language

In MLUI development, the current language is a Workbench Options setting whose purpose is twofold. The MLUI application is prepared using the Current Language setting if language panels or display values exist for the Current Language. AppBuilder opens each new language panel or set display value in the Current Language, so that you can more easily develop large portions of your application.

current system

An entity type in the *Information Model* that represents a non-AppBuilder system interfacing with or replaced by a function.

customer

An entity type in the *Security Information Model* , an instance of which represents a AppBuilder application user who has been identified to the security database and assigned a user ID and password. Each customer is a member of a *customer group* .

customer group

An entity type in the *Security Information Model* that represents a logical collection of *customers* . Security restrictions are applied to customer groups.

D

daemon

A process that executes in the background or provides some service that does not require input or intervention.

data definition language (DDL)

Describes data and their relationships in a database.

data flow

Indicates movement of information or data in a diagram.

data item

The smallest unit of named data that has meaning in describing information.

data link

A connection between a *Window Painter* object and an object in the hierarchy (in the *repository*). For an application's user interface to function correctly, you must use Window Painter to create data links between most Window Painter objects and objects in the hierarchy.
For example, to display the value of the *LAST_NAME* field on one of your application's windows, paint an *edit field* object in the window and then create a data link between that edit field object and the *LAST_NAME* value. Remember that *LAST_NAME* must be in the *view* the window owns.

data model

A logical data map that represents the structure of an enterprise's information. A data model describes the functional dependencies and associations among data items, independent of software, hardware, and machine performance.

data modeling

A discipline that methodically defines the discrete bits of information, such as a customer or transaction, an organization needs to conduct its business. A conceptual data model is refined to a fully attributed logical data model.

data object

A collection of data referred to by a single name that links together the smallest unit of data users need to perform a meaningful business activity.

data object entity-relationship diagram (data object ERD)

A diagram that identifies a set of highly related or coupled entities grouped as a result of partitioning the *logical data model (LDM).* It contains related kernel, characteristic, associative, and intersection entities. The focus of the data object ERD is a specific *kernel entity* that is further defined by the entities around it. Data object ERDs show the relationships between kernel entities.

data store

An entity type in the *Information Model* that represents an existing database or a file a current system accesses.

data type

(1) An object type in the *Information Model* that contains a physical description of data, such as size and format.
(2) A property of a data item that indicates the types of information it can contain.

data universe

The collection of data structures a given *rule* can see, which includes the following views:

- Its input, output, and work views
- The window view of any window it converses
- The section view of any report it converses
- The file view of any file it accesses
- The input and output views of any rule it uses
- The input and output views of any component it uses

A rule can change information in any of these views except information in its own input view or in the output view of any rule or component it uses.

Database Diagram tool

A *Construction Workbench* tool for modeling a database by modifying tables and their primary, index, and foreign keys. See also *relational model*
.

database management system (DBMS)

Software that defines, creates, manipulates, controls, and manages a database.

DBCS

See *double-byte character set (DBCS)* .

DBMS

See *database management system (DBMS)* .

DDE

See *dynamic data exchange (DDE)* .

DDL

See *data definition language (DDL)* .

debugging

The process of finding and fixing runtime errors when the application attempts to perform a prohibited task and logic errors with correct code but not what was intended in Rules Language code. When you run the application, it can be monitored and debugged with the help of the RuleView debugger.

default language

Default language is the language of existing non-MLUI application. For new MLUI applications, the default language is the language of the base objects.

default repository

A *repository* populated with system *object* s that AppBuilder provides, such as system components.

deferred component

A system *component* , in a C application, that does not execute immediately when the system executes a *USE COMPONENT* statement. The component's execution is deferred (or delayed) until the next *CONVERSE WINDOW* statement executes.

delete

An action that permanently removes an *object* from the *repository* and any place it may appear, such as a drawing. See also *clear* .

denormalize

A process in the *Database Diagram tool* that lets you copy a *column* from one table to another through a *foreign key* . Contrast with *normalization*.

distributed application

An application that can be installed and distributed throughout the network across many systems and platforms.

domain

A set of acceptable values that business rules establish for an attribute or field, for example, annual salary must be greater than $10,000 and less than $200,000.

donor entity

An entity that passes one or more foreign keys to a recipient entity. See also *foreign key* and *recipient entity* .

double-byte character set (DBCS)

A set of characters in which each character is represented by two bytes. DBCS enables national language support (NLS), a standard that supports Asian languages with thousands of characters. The AppBuilder environment supports the double-byte character set. See also *single-byte character set (SBCS)* .

download

The process of copying one or more objects from the *Enterprise Repository* on the mainframe to the *Personal Repository* on a workstation. This is also called *refresh* . See also *upload* .

drawing

An object type in the *Information Model* that stores the output of the diagramming tools.

drawing tools

In AppBuilder in the Construction Workbench, these are the tools to design an application that can later be constructed (using construction tools). These tools include:

- Database Diagram tool
- Entity Relationship Diagram tool
- Process Dependency Diagram tool
- State Transition Diagram tool
- Window Flow Diagram tool

driver rule

The *rule* controlling the highest level of the application logic. Often this is the *root rule* .

dynamic data exchange (DDE)

A protocol for moving data among different applications. The AppBuilder environment includes DDE components so you can create AppBuilder applications that can interactively access other Microsoft Windows applications. This is an excellent way to connect third-party workstation applications to mainframe applications and information sources.

dynamic linkage

Allows the AppBuilder runtime system to receive control and either load or locate a load module before passing control to it. You usually employ dynamic linkage to test each module separately in development. In the production environment, static linkage binds the modules together at link-edit time and they then execute as a single module.

E

edit field

A *Window Painter* object that corresponds to a field in the hierarchy (or repository) and in which data is displayed. If the field is not protected (that is, read-only) users can type and modify the edit field's data. See also *data link* .

EJB

See *Enterprise Java Bean (EJB)* .

empty repository

Reference file, with no objects created.

Enterprise Java Bean (EJB)

Container for server-side Java.

Enterprise Repository

A mainframe-based *repository* that contains all the entity types and relationships that describe an organization's information systems.
In a typical AppBuilder development process (unless you are using a *Workgroup Repository*), you must *download* enterprise repository objects that you may be able to reuse to your *Personal Repository* . After creating your application, *upload* the objects (including any new objects you created) back to the Enterprise Repository. You can then use the *migration* process to move your work into a testing and/or production environment.

entity

An object type in the *Information Model* that describes the business data an enterprise uses.
For example, an enterprise that rents automobiles might have entities for *CUSTOMER* and *RESERVATION* . You define entities in the *Entity Relationship Diagram tool* and refine them to file objects at any stage of development. Use *attribute* s to define entities, and relationship objects to describe the relationships between two entities.
See also *object type* .

Entity Relationship Diagram (ERD)

A graphic representation of entities and their relationships with each other. Use the *Entity Relationship Diagram tool* to draw an ERD - also called a *logical data model (LDM)* . Typically, labeled boxes represent entities and lines connecting the boxes are the relationships between them. The *cardinality* of relationships is often expressed in ERDs.

Entity Relationship Diagram tool

A *Construction Workbench* drawing tool for creating a data model of an application or system. You produce an *Entity Relationship Diagram (ERD)* by creating and editing the entities and the relationships between them.

ERD

See *Entity Relationship Diagram (ERD)* .

event

An entity type in the *Information Model* that represents an incident that causes a data object to change states. An event happens at a particular point in time, has no duration, and triggers a set of processes. Events may be an explicit event or implicit event.

event trigger

A drawing object in the *State Transition Diagram tool* that represents the relationship between a process and the event it triggers.

event-driven processing

A processing method that creates and traps system and user events in addition to graphical user interface (GUI) events.

exceptions

Entities outside the rebuild package that rebuilding can affect.

execution environment

The combination of hardware and operating systems on which end users access a completely developed and tested application.
Do not confuse the execution environment and the production environment. AppBuilder applications can execute in both development and production environments. In the development environment, you can move applications between the parts of the Construction Workbench. In contrast, the production environment is only a runtime environment that provides end users access to the application.

exit

A user exit routine that receives control at predefined user exit points.

explicit event

Explicit events are defined within a hierarchy. See also implicit event.

explode

An action in the *Hierarchy window* that displays hidden relationships or objects in the hierarchy. See also *collapse* .

export file

A file that contains information and objects from a repository after the export action. There are two types of export: Full repository export and migration export.

external agent

An object existing outside a system that provides information to or receives information from the system.

F

field

An object type in the *Information Model* that defines the smallest subdivision of data, such as a column in a DB2 table, or part of the input or output definition of other object types, such as the view a window owns. Views organize fields into data structures.

For example, *Customer Last Name* could appear on a window or screen in a physical application, be passed from one rule to another, or be stored in a database.
A logical attribute may become a physical field.

file

An object type in the *Information Model* that represents a physical data file or table on a disk. Rules and components must be related to the file objects that represent the disk files they read and write.

ftp

File transfer protocol. Used in AppBuilder for migrating objects between repositories and for communications between machines. See migration.

focus

The currently active tool, object, or window that receives input from the mouse or keyboard.

forking server

A server technology that invokes a new process for every single client connection that ends when the connection is closed (spawns a child process for each connection request it receives; the child process ends when the connection is closed). Until the connection with the client is closed, the process is dedicated to satisfying service requests from that client. See also *autostart server* and *banking server* .

foreign key

A unique index into another table that can be used to join the two tables. A foreign key is one or more columns that uniquely identifies rows in another table that associates two entities through a relationship.

forward engineering

A systems development method in which a project team first specifies a *logical data model (LDM)* and then derives a physical model from it. The forward engineering process in the *Entity Relationship Diagram tool* translates logical objects (*objects*, *identifiers*, and *attributes*) into relational objects (*tables*, *keys*, and *columns*).

frontier

Signifies the boundary between two different environments, for example, the boundary between the workstation and the mainframe execution environments.

frontier rule

See *remote rule* .

functions

An object type in the *Information Model* that represents the highest-level business function; that is, the purpose of the application. At execution time, functions appear as a desktop icon or menu option that the end-user can access. Functions contain groups of *processes* that subdivide the application. See also *projects* .

G-H

GUI

graphical user interface. For example, in AppBuilder the Construction Workbench provides an application development environment GUI and the Management Console provides a communications control GUI.

hierarchy

The structure that defines the relationship of a AppBuilder application and specifies how the application executes. See also *Hierarchy window* .

Hierarchy window

A window in the *Construction Workbench* that contains the following tabs:

- Project tab - The *hierarchy* and functional diagram of the application
- *Configuration Tab* - The configuration and partitioning information for preparing and deploying a distributed application
- Repository tab - The *hierarchy* and relationship of a specific object within the *repository*
- Inverted tab - An object's inverted hierarchy (showing ownership of a specific object within the application's hierarchy or *repository*)

host repository

See *Enterprise Repository* .

Host Workbench

An interface to the enterprise repository from the mainframe that provides software developers with various tools for generating and testing mainframe-only and host-server application code. See also *rule* and *component* .

HTML

hypertext markup language

I-J

identifier

An object type in the *Information Model* that consists of one or more attributes that uniquely identify an instance of a parent entity or cross-reference another entity. An identifier is a logical key that becomes a physical key during database design.

impact analysis

The first step performed before a migration to determine which modules need to be moved and what effects (if any) the migration will have on other modules. Impact analysis can tell you what must be rebuilt so you do not have to manually trace every relationship the field is connected to. See also *migration* .

implementation name

A *property* type of all *object type* s that uniquely identifies within the operating system each *instance* of that object.
For example, the implementation name of a rule specifies the name of the executable file it generates. Implementation names are required because some environments have special naming constraints.

implicit event

Implicit events are defined externally. Implicit eventing uses a subcell table to identify the routes to the children of the local host, a trigger table to identify the triggering service, and an event table to identify the triggered service, message, or event. The subcell, trigger and event tables are located in AppBuilder\NT\RT. Implicit events are ONLY supported by C client and server.
Implicit eventing is a type of asynchronous processing in which eventing logic resides in text files rather than application code. Implicit eventing can be used for a variety of reasons. For example, one may use it when monitoring the activity of remote services closely in the initial stages of implementing a system but only intermittently after that, in response to unusual network demand. Another example would include wanting to be able to react to a period of market instability by specially logging all trades affecting a particular stock. In both these situations, you'd want to avoid embedding in your application a function that will eventually become superfluous. Implicit eventing addresses the temporary requirements each situation imposes by maintaining eventing logic in text files rather than program code.

IMS

Information Management System, a transaction and hierarchical database management system from IBM.

index key

One or more non-unique columns that can locate more than one of an entity's instances.

Information Model

A collection of object types used to model your business and build a AppBuilder application. These objects and their relationships are the building blocks of your application. Create instances of these objects (for example, rules, windows, fields, etc.) to build the application.

INI file

A text file that contains product configuration or initialization information. It consists of a series of named sections, with each section listing one or more keyword parameters and their assigned values. Each section starts with the section name enclosed in brackets, for example [Java Settings]. Each keyword parameter is associated with the value after the equals sign, such as *OPTIONS=/A/B* .

input view

See *view* .

instance

See *object instance* and *relationship instance* .

Integer

A *data type* that denotes a four-byte integer between -2,147,483,647 and +2,147,483,647.

Interactive System Productivity Facility (ISPF)

An IBM licensed program that is a full-screen editor and dialog manager under the mainframe operating system. ISPF is used to write application programs and generate standard screen panels and interactive dialogues between the application programmer and terminal user.

intersection entity

A type of entity that exists because of the association of two or more kernel entity types. Unlike the *associative entity* type, the intersection entity type does not contain any non-key attributes. The primary way to resolve many-to-many relationships between entities is to create intersection entity types.

inverse scope processing

Process in which rebuild analyze looks at every entity in the rebuild package, whether or not its parent has changed, to determine whether the date it was last changed is more recent than the date it was last installed.

job control language (JCL)

A control language on the mainframe that submits batch jobs to the mainframe operating system.

K-L

kernel entity

A basic kind of entity, such as Customer, Product, and Account, that is independent from any other entity. This independence does not imply that it cannot have a relationship with other objects. A kernel entity must be unique with its own *identifier* and not need relationships with any other entity, although it can have them.

key

(1) One or more attributes that comprise an entity's unique *identifier* . The key structure depends on how the business recognizes the entity.
For example, the *Employee Number* attribute may uniquely define the key for the *Employee* entity.
Different types of keys include: *alternate identifier* , *foreign key* , and *primary key* .
(2) An object type in the *Information Model* that represents a key, which is one or more unique columns that identifies one or more instances-rows in a table-of an entity.
See also *index key* , *foreign key* , and *primary key* .

keywords

A columnar list of words that you can use to associate objects stored in a *repository* .

Language Editor

An AppBuilder tool used to manage repository language objects (used in MLUI development).

leaf process

A process that is defined by only one *rule* (a *root rule*). Another process *never* defines a leaf process, which describes a business activity that comprises a *logical unit of work (LUW)* - in effect, a single application.

line anchor

An object in the drawing tools that indicates the end of a line or a movable area of a box. You can select a line anchor and drag it, changing the size and/or location.

list box

A *Window Painter* object that is a rectangle with scroll bars. A list box can display database records as rows and columns. Users can choose one of the selections in the list box but cannot edit its information. A list box should be linked to a field of a view that occurs more than once. A list box is similar to a *multicolumn list box* but can display only one column (that is, one *field*).

local repository

See *Personal Repository* .

local rule

A rule located in the current environment.

location

An entity type in the *Information Model* that represents a physical business location or category of location at which a procedure is carried out or data is manipulated.
For example: An office, installation, or factory are typical location entities.

logical data model (LDM)

An extension of the conceptual data model, the LDM represents the business area's information needs, focuses on what the business is (rather than what it does), and is depicted in an *Entity Relationship Diagram (ERD)* . Entities in an ERD are related to each other according to business rules. The *cardinality* , relationship names, and text are attached to the relationships. The LDM is an idealized model that is independent of software and hardware.
Entities in the LDM are:

- Assigned the relationships and cardinality between two entities
- Defined with attributes
- Described with text
- Assigned identifiers

See also *Entity Relationship Diagram (ERD)* .

logical process

An entity type in the *Information Model* that specifies an activity that transforms input data. It defines what must be done but not how to do it.

logical server

An object type in the *Information Model* that allows an application to define groups of rules for the purpose of being offered on a server machine. A logical server object has no inherent machine information, but is itself associated with one or more machines using configuration units. See also *Configuration Tab* .

long int

A "long integer", a data type that denotes a four-byte integer between -65,534 and +65,534. Often just simply called "integer" or "int". Contrast with smallint.

logical unit of work (LUW)

A business activity that can be (or should be) performed in isolation and in its entirety.
For example, adding, deleting, and updating a customer record are all discrete business activities, and each constitutes a LUW. On the other hand, accessing a customer file is not an LUW because it is not an independently performed business activity.
A logical unit of work typically consists of a combination of activities, such as:

- Obtaining data for a new customer record
- Accessing a customer file
- Creating a new record
- Populating a new record with data
- Confirming the successful addition of a new customer record

Loopback adapter

This is a tool for testing where access to a network is not feasible, such as on a laptop on which AppBuilder can be installed. This is a third-party tool available from Microsoft.

LUW

See *logical unit of work (LUW)* .

M

Management Console

The AppBuilder program that maintains and coordinates AppBuilder communications and offers a graphical user interface to allow you to view and modify the servers, gateways, and agents involved.

mainframe repository

See *Enterprise Repository* .

mainframe rule

A rule that executes tasks on the mainframe, typically updating corporate data. Although a host rule is not platform specific and can use another host rule, it cannot use a *workstation rule* .

marshalling

A technology for resolving incompatibilities between data definitions across platforms by converting a representation specific to the sending platform to a platform-independent representation and then to a presentation specific to the receiving platform.

Matrix Builder

A *Construction Workbench* tool for creating matrices of relationships between objects in the repository. One use of the Matrix Builder is to create a *CRUD matrix* that maps one kind of entity type against another to show the relationships between entities and the processes that create, read, update, and delete them. See also entity type and *relationship type* .

menu bar

The list of choices that can be selected from a window. Selecting a choice does one of three things: displays a pull-down menu, displays a new menu window, or invokes an application. See also *pull-down menu* and *window* .

Method

An operation upon an object, defined as part of the declaration of a class; all methods are operations, but not all operations are methods. The terms message, method and operation are usually interchangeable. In some languages, a method stands alone and may be redefined in a subclass; in other languages, a method may not be redefined, but serves as part of the implementation of a generic function or a virtual function, both of which may be redefined in a subclass.

migration

A facility that automates promoting software by exporting objects from one *repository* and importing them to another. Between the export and the import, you typically use the Analyze Migration Impact (AMI) facility. After the import, a generated report details what was moved. The migration facility also provides configuration management, which automatically captures and stores the maintenance history of an application's relationships.

Migration Workbench

A tool that lets you move objects from one *repository* to another.

Multi-Language User Interface (MLUI)

The ability to build multiple user interfaces for one application, creating windows and sets in different languages.

modeless window

A type of movable, fixed-sized window that does not require users to take any action, so they can continue uninterrupted with the application.

module

An executable processing block such as a rule or component.

multicolumn list box

A *Window Painter* object that is a rectangle with scroll bars which contains columns and rows of information. A multicolumn list box can display more than one column-that is, more than one field. Users can select one of the choices in the list box and can edit any field that is not protected. A multicolumn list box is linked to fields in a view that occurs more than once.

MVS

Multiple Virtual Storage, the predecessor to OS/390 and zSeries, a mainframe operating system from IBM. The related versions are MVS/XA (extended architecture) and MVS/ESA (enterprise system architecture).

N

name

A property type of all object types that uniquely identifies each instance of that object. You name an object when you create it. Various restrictions and conventions apply to naming objects. See also object type, implementation name, and system ID.

name service

An external service that provides information about network objects, such as the location of servers.

named pipes

An inter-machine communication protocol supported by IBM Microsoft Windows.

national language support (NLS)

The AppBuilder environment supports national languages (including those that require the *double-byte character set (DBCS)*) in two ways:

- You can develop applications in languages other than English (such as German or Japanese)
- The end-user interface can appear in different languages.

However, the *repository* supports only one language at a time.

navigate

To move from an object in one tool to its corresponding place in another tool.
For example, you can navigate from an entity in an entity relationship diagram to the *Hierarchy window* that shows only the objects that entity is concerned with.

normalization

A method of reducing data structures to their simplest possible form, eliminating redundancy while maintaining integrity. The primary purpose of data normalization is to refine a view of entities and attribute relationships, and provide unique access to it. Contrast with *denormalize* .

O

object

Either an object type or a relationship in the *Information Model* . Thus, a *rule* , a *window* , and the *converse* relationship between them are all objects.

object property

The characteristics and properties of an object type. All AppBuilder object types have at least two properties: *name* and system ID.

object instance

An object type a *Construction Workbench* user creates. See also *object type* .
For example, the *ADD_NEW_CUSTOMER* rule is an instance of a rule object.

ObjectSpeak

A set of window-related functions and procedures you can use in your *Rules Language* code. Use the functions to query the properties of windows and their controls (such as push buttons, check boxes, list boxes, and the like). Use the procedures to set the properties or perform some other action.

object type

Represents anything about which an enterprise or system wants to maintain or manipulate data. An object type can be tangible, such as a product or a customer, or abstract, such as a financial transaction or loan agreement. An object type is a discrete element that represents a logical unit of a business system. Relationships store the associations between object types.
The *Information Model* describes the object types, such as rule, window, or component, and the relationship types between them that are available to application designers and developers. Other object types include components, events, functions, processes, rules, states, transitions, views, windows, and so on. See also object.

occur

A property of the includes relationship that determines how many copies of a single view can appear. A view occurs if the property value is more than zero.

organization

An entity type in the *Information Model* that represents a logical unit or group, such as a department or division, that carries out a function in an enterprise.

orphan

An object in a hierarchy display that has no parent. Also called the root object.

output view

See *view* .

P-Q

package

A logical grouping of Objects (Classes). Packages are commonly used to organize classes belonging to the same category [project, application, etc.] or providing the same functionality

partition

An object in AppBuilder that defines that a segment of an application is to be prepared to (built on) a specific machine or platform type.

PDD

See *Process Dependency Diagram (PDD)* .

Personal Repository

An AppBuilder *repository* that resides on a workstation and contains the entity type and relationships that describe the application you are building. When you design and develop an AppBuilder application, you typically begin by downloading *Enterprise Repository* objects to the personal repository that you may be able to reuse. You finish by uploading your work, which includes new objects you have created.

pilot

A small application developed to evaluate the effectiveness of a particular design technique before undertaking production on a larger application.

prepare

To ready (by compiling, building, etc.) an object or application for execution, to create executable files from the source code and hierarchy. See also *Super Prepare* .

primary key

A required key consisting of one or more unique columns that identify a single instance-a row in a table-of an entity type.
For example, *EMPLOYEE_NUMBER* could be the primary key of the *EMPLOYEE* entity and *CUSTOMER_ACCOUNT_NUMBER* the primary key of the *CUSTOMER ACCOUNT* entity.

processes

(1) An object type in the *Information Model* that describes business activities that comprise a logical unit of work. Each process represents a single application (leaf process), or a set of applications. You convert your hierarchy of processes into an end-user menu or icon.
(2) A *Process Dependency Diagram (PDD)* drawing object that represents an activity that transforms input data into output data.

Process Dependency Diagram (PDD)

An AppBuilder diagram that illustrates the dependency of business activities (processes) on each other and the events that trigger their initiation. The PDD shows a sequenced set of actions that occur in response to a given event stimulus. Process dependency modeling is a way to identify and document internal, external, and time-delayed events that affect the business.

Process Dependency Diagram tool

An AppBuilder drawing tools available in the Construction Workbench for creating a Process Dependency Diagram (PDD).

process trigger

A relationship in a *Process Dependency Diagram (PDD)* that connects an event and an action process or decision process, and that indicates when processes begin.

production environment

The runtime environment in which end users access a completed application.
Do not confuse the production environment and the execution environment. AppBuilder applications can execute in both development and production environments. In the development environment, you can move applications between the parts of the Construction Workbench. In contrast, the production environment is only a runtime environment that provides end users access to the application.

productivity

The amount of goods or services produced per unit of labor or expense. This equates to the number of function points divided by the amount of labor devoted to the project.

projects

(1) User management information (that is, user, group, project, etc.) in a *repository* .
(2) In a AppBuilder application, the project contains the business *functions* and associated *application configuration* (s).

property

A characteristic defining a particular instance of an object, usually defined in the Properties window for the object. An object or relationship's name, system ID, data type, or range of possible values are all properties. See also *object property* .
As a part of a class, property is a variable declared in the class available for all the objects created in that class.

protocol

A set of rules or transport mechanisms that describe communications.

prototype

A model or preliminary implementation suitable for clarifying requirements or for evaluating a system's design, performance, and production potential. Prototypes are to demonstrate the look and feel of an application's user interface.
A prototype contains just enough code to show users the application's:

- Menu structure
- Navigation path between windows
- Pull-down menus that initiate leaf processes
- Reports

pull-down menu

A list of choices that users can display by selecting a choice on a menu bar.

purge

To remove objects from the personal repository without flagging them for uploading to the mainframe.

push button

A *Window Painter* object that is a small rectangle with rounded corners. It is not linked to an object. When users select a push button, it returns control to the rule that invoked the window and usually initiates an immediate action, such as further processing.

R

radio button

A *Window Painter* object that is a small circle and indicates an on/off or yes/no condition. A black dot fills the center of the circle when users select a radio button. A group of radio buttons within a group box indicates mutually exclusive choices. That is, users can select only one radio button in the group because all the radio buttons in the group are linked to the same field.
See also *data link* .

recipient entity

Receives one or more foreign keys from its donor entity. A *cardinality* relationship determines whether an entity is a donor or a recipient. See also *donor entity* and *foreign key* .

recursive

A module that can invoke itself. Use recursive modules with extreme caution.

recursive relationship

A relationship an entity or state has with itself. Recursive relationships are shown graphically with a line that loops from the entity or state onto the same entity or state (itself).

reentrant

Code written so that it can be shared by several programs at the same time. Only one copy of the operating system routine needs to reside in memory to serve all executing applications.

refresh

To download one or more objects from the *Enterprise Repository* on the mainframe to the *Personal Repository* on the workstation. See also *download* .

relational model

Expresses data structures and relationships in a table. In AppBuilder applications, a database diagram represents the relational model that results from translating the logical objects in an *Entity Relationship Diagram (ERD)* (entity, identifier, and attribute) to relational objects (table, column, and key). See also *Database Diagram tool* .

relationship

An object type in the *Information Model* that records information about the relation between objects (even between relationships).
For example, if you have two *entity* object types, *RESERVATION* and *CAR_TYPE* , you might have a relationship object called *SPECIFIES* to describe how they are connected.
An attribute object describes the relationship between objects (entities). Do not confuse the relationship object with relationship, which is the relationship between any of the other object types in the Information Model.

relationship instance

A relationship that a *Construction Workbench* user creates.
For example, the converse relationship between the *ADD_NEW_CUSTOMER* rule and the *CUSTOMER_DETAIL* window is an instance of a converse relationship.

relationship property

The characteristics of a relationship. Some relationships in the *Information Model* have attributes; in traditional entity-relationship modeling, relationships usually do not have attributes.

relationship type

A connection, association, or link between two object types.
For example, an instance of the include relationship type between the *CUSTOMER* and *ADDRESS* entities is:
CUSTOMER includes one or many *ADDRESS* (s)
Cardinality is the numerical condition a relationship has between its entities (in the example above, one or many).
Each relationship has a label. For example in the Information Model, the relationship between a rule object and a window object is a converse relationship. Some relationships in the Information Model have attributes; in traditional entity-relationship modeling, relationships usually do not have attributes. Do not confuse a relationship type with the relationship object, which is a particular kind of Information Model object type.

remote preparation

When compilation of a program is initiated on the development machine but is transmitted to another machine, where it actually takes place.

remote procedure call (RPC)

A technology for requesting services on remote platforms. Programs that use remote procedure calls execute synchronously: clients request a service, wait for a response, and continue processing only on receiving it.

remote rule

A rule that is to be called from the current executing rule as a server rule using communications. Usually a remote rule is located on a different machine than the rule calling it. Also known as a frontier rule, a remote rule is an entry point into a logical server from a client. Remote rules are attached directly to one or more logical servers in the Configuration hierarchy (Configuration tab of the Hierarchy window).

remove

To remove objects from the Personal Repository and flag them for deletion from the Enterprise Repository when you upload. See also Enterprise

Repository, Personal Repository, and upload.

report

An entity type in the Information Model that defines, in conjunction with the section entity type, the paper output an application (usually, a batch application) produces for an end user. You create a report and its associated sections using the Report Painter tool and Report Writer.

Report Painter

This part of the AppBuilder product lets you control the layout and content of custom reports. After the layout is designed, you can then use Report Writer to create reports on the workstation that print on the mainframe.

Report Writer

This part of the AppBuilder product creates custom reports on the workstation that print from the mainframe or Web application servers with the Java Report Writer after you have designed the layout of a report with Report Painter.

repository

An organized body of information that can support business and data processing activities. See also *Enterprise Repository* , *Workgroup Repository* , and *Personal Repository* .

repository administration

Utilities for managing a repository. Use the Repository Administration tool to create, and edit, select, and delete repositories. See also *Enterprise Repository* , Workgroup Repository, and *Personal Repository* .

Repository Administration tool

A tool used to manage Personal and Workgroup Repositories. With the Repository Administration tool, you can compare versions, copy the contents from one repository to another, and manage data migration between separate repositories.

reuse

The practice of using existing objects when possible, rather than creating new ones. See also code reuse analysis (CRA).

reverse engineering

A process in the Database Diagram tool that converts a database diagram's relational objects (tables, keys, and columns) back to the ERD's logical objects (entities, relationships, attributes, and identifiers).

reverse trace analysis

Tracks and reports how the relational objects in the database model correspond to the logical objects in an Entity Relationship Diagram (ERD).

rollback

Abandons any changes you made in a workbench since the startup or the last *commit* or rollback.

root process

The process immediately beneath a function in an AppBuilder application hierarchy. In an AppBuilder application, functions refine into *processes* .

root object

An object in the hierarchy that has no parent. Also called an orphan object.

root rule

A *rule* defines a leaf process. A root rule has no input or output views and is the rule that a process invokes. See also *drawing tools* .

rule

An object type in the *Information Model* that defines the logic of a process, controls the execution of other rules and components, converses windows and reports, and accesses files. In contrast, a component generally performs only a special system task. *Rules Language* statements specify the processing logic. See mainframe rule and workstation rule.
Also, see frontier rule and remote rule and root rule and subrule.

Rule Painter

A *Construction Workbench* tool for writing *Rules Language* source code. Rule Painter is an intelligent editor that helps you write rules quickly and correctly by prompting you with names of fields, views, and other objects in your application's rules hierarchy.

Rules Language

The pseudocode for coding the logic for AppBuilder applications. The syntax and semantics are similar to but simpler than other high-level, block-structured programming languages, such as COBOL and C. Rules Language constructs are translated into any supported target language.

RuleView

A source level debugger available on both the mainframe and workstation that enables you to directly debug *Rules Language* code. RuleView lets you step through rules code line by line, trace through rules, and set *breakpoints* to interrupt execution. Data is presented in a view, and you can collapse, expand, or modify the data in any view that the rule you are currently executing can access.

S

SBCS

See *single-byte character set (SBCS)* .

scope

Controls the possible connections when you navigate between objects. A number of object types comprise a scope, so that only those object types are considered. Scopes are used to control downloading, uploading, and repository reports.

scroll box

A small box in a window's scroll bar that shows the position of the visible information relative to the total amount of available information. Users move the scroll box with a mouse to scroll through the information.

section

An entity type in the Information Model that, in conjunction with a report entity type, defines the paper output an application (usually a batch application) produces for end users. Each section defines a particular part of a report (such as a header section and footer section). You typically create a report and its associated sections using the Report Writer. A property of the Contains relationship determines which part of the report each section defines.

Security Information Model

Defines the object types (users, groups, products, etc.) and relationships between them in the *security system* . This is a subset of the Information Model.

security system

Controls access to production systems in the runtime environment.

seed

The connection between a Migration entity or a Rebuild Package entity to an application.

server

A unit that provides shared services to client workstations over a network.

service

Code that performs some task, such as accessing a database, for a *client* program.

service agent

See *agent* .

session

1. A user's connection to the repository, particularly used to describe connections to a *Workgroup Repository* . Sessions are usually automatically opened and closed by a workbench.
2. Current working time between starting and stopping the Construction Workbench, or between the times you *commit* to the *repository* changes made to the application.

set

An object type in the *Information Model* that represents a group of symbols and the corresponding values. (An older style of set, still supported, used hard-coded values rather than symbols.) See also *symbol* .

sibling

A child object that shares a parent with another such object. That is, the two objects reside at the same level in a hierarchy.

single-byte character set (SBCS)

The standard ASCII or EBCDIC character sets in which a different one-byte code represents each of 256 possible characters. See also *double-byte character set (DBCS)* .

smallint

A "small integer", a data type that denotes a two-byte integer between -32,767 and +32,767. Contrast with long int.

smart scope

An extension to basic scope that provides additional navigational information so you can specify navigational closure. For example, when navigating through the components associated with a rule, rather than all components associated with the rule, smart scope lets you navigate through only those components not provided in the default repository. See also *scope* .

SQLJ

IBM, Informix, Oracle, Sun Microsystems, Sybase, Tandem, and others have contributed to this standard for embedding static SQL statements and constructs in Java programs (ANSI X3.135.10-1998). This reduces development and maintenance costs of Java programs that require database connectivity by providing a mechanism to use SQL statements directly.

stand-alone application

An application that is fully developed and run on one machine, independent of any other system.

state

An entity type in the *Information Model* that is a discrete set of attributes, values, and relationships a data object holds at a point in time. A logical stage in the life of a *data object* , a state has duration, occupies an interval of time, and is often associated with a continuous activity (such as invoice collection). An *event* separates two states, and a state separates two events.

State Transition Diagram

A network diagram of states and events; it shows changes in the behavior of a data object. A State Transition Diagram is a model that identifies the stages (or states) a data object goes through in its life, and the external and internal business events that trigger a set of processes and cause the data object to move from one state to another. A State Transition Diagram constitutes a data object's control view, because it captures business rules that translate into system controls. A State Transition Diagram is also called a state transition model.

State Transition Diagram tool

A *Construction Workbench* tool for producing *State Transition Diagram* .

static linkage

Passes control directly from one object to another without the intervention of the runtime system. All the reports, rules, and components of a statically linked application are bound together at link-edit time and loaded together as one module at runtime. See also *dynamic linkage* .

static text

A *Window Painter* object with a data link that appears on an end-user window as descriptive text-that is, a label.

structural model

Data structures composed of AppBuilder structural objects (files, views, and fields) that an AppBuilder rule uses to read from and write to database tables. You can translate a relational model to a structural model with the *Database Diagram tool* 's transformation process. See also *field* , *file* , *transformation* , and *view* .

Structured Query Language (SQL)

A language used to access relational database management systems.

subrule

The child of a parent rule or a rule that is called from another rule. This is an AppBuilder short-hand way of describing a rule that is called by another rule. In some contexts, it is referred to as the *called rule* as opposed to the *calling rule* .

subscript

The occurring view's argument that specifies a unique occurrence.

Super Prepare

Prepares an object and all of its children.

supertype-subtype hierarchy

Indicates an entity type that shares many of the same attributes as another entity type. Supertypes and subtypes hierarchically structure related data. The supertype has a unique *identifier* and common attributes that all of its subtypes inherit. Each subtype has special characteristics that further classify it as a child to the supertype and differentiate it from other subtypes. Thus, a supertype entity type is a parent entity to its children subtypes.
For example: *Customer* is a supertype classified into *Retail Customer* and *Institutional Customer* subtypes.

symbol

An entity type in the *Information Model* that represents the name of a particular value within a *set* .

symbolic value

See *value* .

system component

These are AppBuilder supplied components (small programs) that can be used to achieve things that Rules cannot. Developers can also writer their own components called User Components. Also refer to user component.

system ID

A property of an AppBuilder object that uniquely identifies it in the repository. A system ID is automatically generated when you create the object and is required because a short, unique name is required to identify objects within the repository. (When created on the mainframe, it has 6, 7, or 8 characters; on the workstation it has 7 characters.)

T

table

A logical data structure in a relational database system. One of the three objects in a relational model, a table is a logical grouping of data into rows and columns, and often implements an entity. See also *column* , *key* , and *relational model* .

TCP/IP

An inter-machine communications protocol supported by IBM OS/2, Microsoft Windows, and the various versions of the UNIX operating system.

thick client

The self-contained part of an application that runs on a client machine without the help of the part running on a server. See also *thin client* .

thin client

The part of an application that runs in a Web browser with information from a servlet or other services running on a server. See also *thick client* .

TP monitor

Software that controls transaction processing across a network.

trace analysis

Tracks how the *logical data model (LDM)* corresponds to the relational database model so you can assess the impact of changing the logical model. You do a trace analysis from the *Entity Relationship Diagram tool* in the *Construction Workbench* . Trace analysis generates a report that shows how logical objects in an entity relationship diagram (ERD) are converted to relational tables. See also *Entity Relationship Diagram tool* , *logical data model (LDM)* , *relational model* , and *table* .

traceability

Additional information stored with *Information Model* objects that enable trace analysis to track how logical objects are translated to relational objects and vice versa. See also *reverse trace analysis* and *trace analysis* .

Transaction ID page (TIP)

Records in a *Workgroup Repository* the identity and statues of currently active connections for data integrity and recovery.

transformation

An engineering process available in the Database Diagram tool of the *Construction Workbench* that transforms a relational model's objects (tables, columns, and keys) to an AppBuilder structural model (files, view, and fields).

transition

An entity type in the *Information Model* that represents an event and the processes it triggers to cause an entity to change states. In a *State Transition Diagram* , a transition is a relationship that shows how a data object can move from one state to another state. A line between two states represents the business event that causes the transition. See also *data object* , *event* , *relationship* , *state* , and *State Transition Diagram* .

U-V

Unit of Work (UOW)

Unit of Work is a group of only your changed objects. Using a UOW simplifies the upload, download, and migration processes. See logical unit of work (LUW).

upload

The process of copying an object from the *Personal Repository* on the workstation to the *Enterprise Repository* on the mainframe. See also *download* .

user component

User Component is a user written program that achieves something that AppBuilder system components and Rules cannot. See also component.

value

An object type in the *Information Model* that represents any constant.

version

An object in the repository that represents a set of individual objects. A version object acts as a view of the repository. It sees and manages a set of objects, their relationships, and object revisions. A version sees specific revisions of objects and relationships.

version states

The stages of a version. There are four allowable states: Working, Text, Frozen, and Released. The state of a version can be changed to reflect its use.

view

An object type in the *Information Model* that defines data structures in AppBuilder applications. A view is a group of properties that further describes and characterizes an object. Views define input and output data structures of a rule, a component, a window, a file, and a section. Views include other views and fields, which can create very sophisticated data structures.

W-X-Y-Z

window

An object type in the *Information Model* that defines the interactive user interface to an application. You paint objects on windows to let end users view data in different formats, modify data, or control application execution. These objects include check boxes, combo boxes, edit fields, list boxes, multicolumn list boxes, push buttons, radio buttons, menus, and static text.

Window Flow Diagram tool

A *Construction Workbench* tool that lets you model window flow and user interaction during the prototyping phase.

Window Painter

A *Construction Workbench* tool for designing and developing an application's end-user interface. Using window, view, and field objects defined in the *repository* as input, Window Painter enables you to control the size, shape, and colors of window, determine the layout of the fields, define push buttons and menu selections, and so forth. You determine what the end user can see and do in each window.

window view

A view linked to a window object instance.

work view

A property of the *owns* *relationship type* relationship that defines a view as a temporary local storage area visible only to the owning rule. A *rule* can read from or write to its own work view but cannot see any other rule's work view. Only a rule can own a work view.

Workgroup Repository

An AppBuilder *repository* that developers access through a network. Their work is concurrent and other developers within their group can access it without importing or exporting the data. (See also Workgroup Repository.)

workstation

An object type in the *Security Information Model* , an instance of which represents an individual workstation. Each workstation can be a member of a single *workstation group* .

workstation group

An object type in the *Security Information Model* that represents a logical collection of *workstation* s. Security restrictions are applied to workstation groups.

workstation rule

Any *rule* that executes on the *workstation* . Workstation rules are not platform specific and can use mainframe rules. Examples of workstation tasks that workstation rules execute include drawing windows and displaying data. (A rule that IS platform specific is a mainframe rule.)

XML

Extensible markup language. Used in parts of AppBuilder.