Oracle® Insurance IStream

IStream Author User Guide

Release 6.2

E14878-01

January 2009

ORACLE

INSURANCE

Copyright

Primary Authors: Andrew Brooke and Ken Weinberg

CONTENTS

Chapter 1

Overview

Welcome to the IStream Author User Guide.

IStream Author is the tool you use to create, edit and manage model documents.

IStream Document Manager, a related application that you can use with Author, automates document assembly, management and control. Documents can range from complex, multi-page, multi-format contracts, to statements with extensive customization, and even simple correspondence.

This chapter includes the information about:

Document Conventions

Tips, Notes, Important Notes and Warnings

Tip: A **Tip** provides a better way to use the software.

Note: A **Note** contains special information and reminders.

Important: An **Important** note contains significant information about the use and understanding of the software.

Warning: A **Warning** contains critical information that if ignored, may cause errors or result in the loss of information.

Other Document Conventions

- Window names, buttons, tabs and other screen entities are in bold, for example: Click **Next**.
- Paths, URLs and code samples use the Courier font, for example: `C:\Windows`
- Some sections contain links to other **Related Topics**.

Overview of IStream Document Manager

IStream Document Manager is a suite of integrated document capabilities, including document assembly, document management, search, workflow, collaboration and customization that supports the entire document life cycle across the enterprise. It enables organizations to better create, manage and distribute highly personalized, complex, error-free documents such as contracts, booklets and correspondence.

IStream Writer, IStream Author, IStream Author Add-in for Microsoft Word and IStream Visualizer are available with IStream Document Manager to assist with the process of creating model documents. Model documents are used to define the business rules, text, variable information, graphics and dynamic charts that are used to create your personalized documents. Below is a brief description of each:

IStream Writer

IStream Writer facilitates the product development process for creation of state specific documents for filing purposes. It is also used to create specifications for the documents so that when they're imported into IStream Author, the specifications automatically create model documents that reflect all of the rules, text, and graphics identified with IStream Writer. When used in conjunction with IStream Author, Writer can significantly reduce the effort and complexity of creating model documents, shortening the creation process from weeks to days.

IStream Writer is an optional component of the IStream suite. For further information on IStream Writer, refer to the *IStream Writer User Guide*.

IStream Author

IStream Author is the primary application used for the creation of IStream model documents. Microsoft Word is leveraged within the IStream Author interface. Author provides all of the functionality needed to address creation of both simple and complex documents. It allows you to create the rules, format the text and identify the data that will be used in creation of the documents. It provides wizards, the ability to set up key data, and the option of using the model document outline. With IStream Author, you can fully test your documents using the same engine that is used for production.

This manual introduces the features available in IStream Author, and shows you how to those features to create your documents.

IStream Author Add-in for Microsoft Word

IStream Author Add-in for Microsoft Word allows you to create and test IStream model documents. It is an add-in for Microsoft Word that provides the majority of the functionality needed to create both simple and complex documents. It allows you to create the rules, format the text and identify the data that will be used in creation of the documents. It provides maximum flexibility for leveraging Microsoft Word features such as print preview and macros. It does not include access to Author wizards, key data setup or model document outlines. IStream

Author Add-in includes the ability to fully test your documents using the same engine that is used for production.

For further information on IStream Author Add-in for Microsoft Word, refer to the *IStream Author Add-in for Microsoft Word User Guide*.

IStream Visualizer

IStream Visualizer is used for the dynamic creation of charts and graphs. It supports line, pie, column and bar charts and graphs. IStream Visualizer is used from within IStream Author.

IStream Visualizer is an optional component of the IStream suite. For further information on IStream Visualizer, see *Charts and Graphs* on page 211.

IStream Document Manager Components

IStream Document Manager consists of client components that are usually located on your workstation or desktop, and server components which can be installed on one or more separate computers somewhere on your network.

Client Components

The following client components are included with IStream Document Manager:

- IStream Author
- IStream InfoConnector
- IStream Assembler
- IStream Author Add-in for Microsoft Word
- IStream Customizer
- IStream Promoter
- IStream Visualizer (optional module)

Server Components

You use the following server components with IStream Author:

- IStream Document Manager Integration
- IStream Assembler (server version)

IStream Document Manager Documentation

IStream Document Manager includes the following documents and online help files. If you need a copy of any of these documents, please contact your system or product administrator.

General Documentation

- The *IStream Document Manager Release Notes* include general product information, product enhancements and new features, supported platforms and third-party software, assorted considerations, and known issues and limitations.
- The *IStream Document Manager ReadMe* file describes the contents of the installation CD.
- *IStream Document Manager*The *IStream Document Manager Glossary* contains definitions of commonly used IStream terms.

User Guides and Online Help

- The *IStream Assembler Online Help* describes how to generate documents from sections and model documents.
- The *IStream Assembler Add-in for Microsoft Word Online Help* describes how to perform certain IStream Assembler functions directly within Microsoft Word.
- The *IStream Author User Guide* describes how to create, edit, and test sections or model documents.
- The *IStream Author Add-in for Microsoft Word Online Help* describes how to perform certain IStream Author functions directly within Microsoft Word.
- The *IStream Customizer User Guide* describes how to modify the content of generated documents so that the changes will be applied during subsequent document generations.
- The *IStream Document Manager DMS Guide for IStream and Model Documents* describes how to edit and work with IStream sections and model documents using the DMS user interface.
- The *DMS Plug-in for Author Online Help* describes how to access DMS functionality in IStream Author.
- The *IStream InfoConnector Online Help* describes how to set up and configure IStream InfoSources.

Advanced Guides

- The *IStream Document Manager Guide for New Installations* explains how to complete a new installation of IStream Document Manager. It includes system requirements and detailed installation and configuration information.

- The *IStream Document Manager Upgrade Installation Guide* explains how to upgrade an *existing* installation of IStream Document Manager. It includes system requirements and detailed installation and configuration information.
- The *IStream Document Manager Message Reference Guide* contains lists of error, log and SDK messages from the various IStream components. It is for technical users who need additional information about the various IStream messages they receive.
- The *IStream Document Manager Technical Guide* is for system administrators and technical support staff who configure IStream Document Manager and set up links between it and their company's database. This guide is also for those who set up and maintain security groups and operators in the system, and who install, optimize, maintain and troubleshoot IStream Document Manager. It also describes installing and configuring IStream XML InfoSources for interactive and batch generation.
- If you have purchased the IStream Toolkit, the *IStream Assembler* and *Extensibility Toolkit Guides* are installed onto your system. (You can view these guides from the **Start > All Programs > IStream** menu.)

  These guides contain detailed descriptions of the components in the IStream Toolkit, or SDK. They are for technical users who need to integrate IStream components with their own or other third party applications.

  In addition to the toolkit guides, the *IStream Toolkit SDK Samples Guide* gives an overview of the toolkit samples. The samples are working examples that can help you develop custom applications.

  For more information about the toolkit, see the *IStream Document Manager Toolkit ReadMe*.

Using the Online Help

This section describes how to use the Online Help and includes information about:

The Contents of the Online Help

The Online Help contains the same contents as the related PDF document, but in an online Help format.

To open the Online Help, click the **Help** menu**.**

The Help is divided into two frames:

- the left frame displays the navigation tools: **Contents**, **Index** and **Search**
- the right frame contains the contents of each Help topic

There are different ways to find a Help topic:

Searching the Help

You can search the entire Help contents to find a specific topic.

Method: Search the Help

---

1. In the left pane of the Help, click the **Search** tab.
2. Enter the word(s) you want to search for, then click **Go!** or press Enter.
3. A list of Help topics is displayed in descending order by **Rank**. The Rank indicates how many times the word(s) you searched for appears in a Help topic. It can help indicate how relevant the topic may be in your search.

Tip: Use specific words in your search, for example: *model document*. Avoid using plurals, for example, "*sections,"* because this may limit your search results.

Using the Help Index

The Help **Index** contains a listing of all the Help topics in alphabetical order.

Method: Use the Help Index

---

1. In the left pane of the Help, click the **Index** tab.
2. Click the letter that corresponds to the topic you are searching for. You cannot select a letter that is greyed out, because it contains no index entries.
3. A list of all index entries beginning with the letter you selected is displayed.
4. Scroll to the index entry of the topic you are searching for.
5. Click the topic to view its contents in the main body of the Help.

Using the Help Table of Contents

When you open the Help, the **Contents** are displayed. The **Contents** contain main topics and their subtopics.

Each main topic appears as a book icon:

Each subtopic appears as a page icon:

Subtopics can also appear as book icons. In other words, books can appear within other books.

You can open a book by clicking a book icon or the text next to the book icon. This will expand the book and display the topics within that book.

To close an open book, click the book icon. The book “collapses”, hiding the topics within the book.

Tip: When a Help topic is displayed, you can click the “Show in Contents” button to open the corresponding book that contains the displayed Help topic:

Navigating the Help

To go to the next or previous Help topic in the **Contents**, use the Next and Previous buttons in the right pane of the Help:

To go to the next or previous topic that you have viewed, use the **Forward** and **Back** buttons in your Web Browser.

Printing a Help Topic

You can print a Help topic in case you want to refer to it later.

Method: Print a Help topic

1. Click the Print icon in the upper-right corner of the Help:

2. The Print dialog box is displayed.
3. Click **Print** to print the Help topic.

Contacting Skywire Software for Help

Customer Support hours are 8:00 A.M. to 8:00 P.M. (Eastern Time), Monday through Friday. Outside of these hours, send us a detailed e-mail message and you will be contacted during regular business hours. Please provide detailed information, as described in the *Support Checklist*.

Contact Information

Mail: Customer Support
Skywire Software
19 Allstate Parkway, Suite 400
Markham, Ontario, L3R 5A4

Phone: 1-905-513-7466

Fax: 1-905-513-1684

Email: directsupport@skywiresoftware.com

Web: www.skywiresoftware.com

Support Checklist

When contacting Skywire Software Customer Support, please provide the following information:

- Your name, company name, e-mail address, and phone number
- Version numbers of all your Skywire Software products
- Name and version of the network software
- Windows version, including any installed Service Packs
- Microsoft .NET Framework version
- DMS version, including any installed Service Packs (if applicable)
- Microsoft Word version (if applicable)
- Database vendor and version (if applicable)
- Error messages and the circumstances of their occurrence
- A full description of the problem:
    - What happened? What were the sequence of events that preceded the problem?
    - In which screen or window did the problem occur?
    - Was the problem the result of pressing a key?
    - Did the screen freeze? What functions of the software are affected?
    - How many people are affected?

Chapter 2

Using and Configuring IStream Author

You use IStream Author to create, edit and manage model documents. When working with model documents, you generate and maintain documents in IStream Document Manager or in file system repositories.

This chapter describes:

The IStream Author Interface

IStream Author's user interface is similar to Microsoft Word's. Because it embeds Microsoft Word within its main window, IStream Author gives you access to most of Word's menus when editing documents.

Method: Open IStream Author

- Click **Start** > **Programs** > **IStream** > **Author**

Note: When you first log in to Windows, an IStream cleanup utility briefly appears in a DOS window.

In IStream Author, there are three windows available for creating your model documents:

- the **Outline or Tree View window** on the left of your screen displays the hierarchy of the model document, showing its master section and all included sections
- the **Editor window** is located in the middle of your screen: this window maintains the contents of the model document, for example the text and code
- the **Compile/Generation Log window** is on the bottom of your screen and shows the compile log, along with any error messages found during compilation

Note: You can open and close the **Outline** and **Compile/Generation Log** windows by clicking their respective toolbar buttons. You can also select and drag these windows to any side of the **Editor** window.

IStream Author has additional menu items beyond Word's menu items for IStream functions. When IStream Author is first opened, you can see the **File**, Author, and **Help** menus. When you open a model document or section in IStream Author, these menus merge with Word's menus to provide the functionality you need from Word to create and edit model documents.

Note: Livelink menu functionality (installed with the Livelink Explorer module) is not supported.

Related Topics

- *Accessing Word Functions* on page 23
- *Configuring Auto Formatting Features in Word* on page 24

Accessing Word Functions

When IStream Author embeds Word in its interface, it takes control of the **File**, **Window**, and **Help** menus. Some of the items in these menus, previously available in Word, are no longer available because of the controls IStream Author uses. However, you can still access these items by selecting **Tools** > **Customize** in Word. You must make these changes in Word *before* you start IStream Author for them to take effect in IStream Author. For an example of how to add Word functions, see *Adding Page Setup to the IStream Author Toolbar* on page 23.

Related Topics

- *The IStream Author Interface* on page 22
- *Configuring Auto Formatting Features in Word* on page 24

Adding Page Setup to the IStream Author Toolbar

The following procedure is an example of how to add Word functions to the toolbar in IStream Author.

1. Ensure IStream Author is closed.
2. Start Word.
3. In Word, do one of the following steps:
   - select **Tools > Customize**, and click the **Commands** tab
   - in the **Categories** list, click **File**
   - in the **Commands** list, scroll down to **Page Setup**
   - click the **Page Setup** button and drag it to a location on the Word toolbar: Word will place the **Page Setup** button on your toolbar; you can drag and move it, or drag it off the toolbar to delete it

Repeat this procedure for any other menu item or command you want available on the IStream Author toolbar.

When using a specific Word template as the basis for all your model documents and sections (the recommended practice), ensure that you apply any customizing of the toolbar buttons to the template as well. At the bottom of the **Customize** dialog box is the **Save Changes** option containing a drop-down list of templates. The default is **Normal.dot**. See the Word documentation for more information on customizing Word.

Configuring Auto Formatting Features in Word

Word maintains default auto-formatting features that can affect the generation of model documents in both the IStream Author and End User Workstations. Therefore, you must ensure that these features are disabled.

For detailed instructions on how to disable these features, see *Disabling Auto-Formatting in Word* on page 24.

Related Topics

- *The IStream Author Interface* on page 22
- *Accessing Word Functions* on page 23

Disabling Auto-Formatting in Word

Method: Disable the auto-formatting features in Word

You need to complete the following method to produce consistently formatted documents.

1. From the **Tools** menu.In Word, select **AutoCorrect Options**
2. Click the **AutoFormat As You Type** tab.
3. Clear the **“Straight” quotes with “smart” quotes** and the **Internet and network paths with hyperlinks** check boxes.
4. Click the **AutoFormat** tab.
5. Clear the **“Straight” quotes with “smart” quotes** and the **Internet and network paths with hyperlinks** check boxes.
6. Click **OK**.
7. From the **Tools** menu, celect **Options**
8. Click the **Save** tab.
9. Ensure the **Allow fast saves** check box is cleared.
10. Click **OK**.

    The auto-formatting features will now not affect the generation of model documents.

Note: Remember to close Word before you launch IStream Author.

Related Topics

- *Configuring Auto Formatting Features in Word*

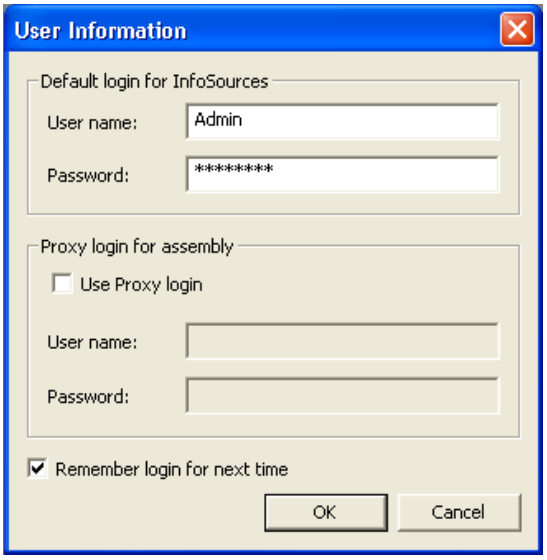
Viewing User Information

The **User Information** dialog box contains user and proxy information that is used whenever IStream Author interacts with InfoSources, Assembler, or other parts of IStream Document Manager. Changing the information here will change it for all IStream applications that use the same information, including the IStream Author Add-in for Microsoft Word, the IStream Assembler Add-in for Word, IStream Author, and IStream Customizer.

Method: View or change user information

1. Click **Author** > **User Info**.

   The **User Information** dialog box opens.



2. To change your default login for InfoSources, enter a new **User name** and **Password**.
3. If you need access to databases with security features, you may need to select **Use Proxy login** and then enter a valid proxy **User name** and **Password**. If you don't know these, consult your system administrator.
4. To have the system retain this login information the next time you log in, select **Remember login for next time**.

Method: Add user information

1. Enter your user information (including your **Proxy ID**).
2. Select the check box next to **Use Proxy login** and enter your Proxy **User ID** and **password**.

3. Select the check box next to **Remember login for next time** to bypass the **User Information** dialog box the next time you open IStream Author. If you leave this box cleared, the **User Information** dialog box automatically opens each time you access IStream Author.

Note: The **User Information** can be entered or changed at any time in IStream Author. To do this, select **UserInfo** from the **Author** menu, and add or change the entries as necessary.

IStream Author Windows

This section describes what you will see while working with IStream Author, and how to use the windows to make the most of your authoring sessions.

The different tools available for viewing in IStream Author are:

- *The Editor Window* on page 27
- *The Outline Window* on page 28 (also called Tree View Window)
- *The Compile/Generation Log Window* on page 28 (also called Log Window)
- *Toolbars* on page 28

The Editor Window

The **Editor** window looks just like the **Editor** window in Word. It is where you add your text and graphics and code for your model documents.

Note: If you have more than one model section open at a time, you can toggle between them by selecting the document from the **Window** menu.

You can close the **Outline** and **Compile/Generation Log** windows if you want more space on your screen to view the **Editor** window. You can restore the **Outline** and **Compile/Generation Log** windows by using the following toolbar buttons:

- **Toggle Outline**: 
- **Toggle Compile/Generation Log**: 

You can also hide the toolbars and status bar so that more of your screen is available for the open document windows. Hide or show the IStream Author toolbars by choosing **View** from the **Author** menu. Hide or show **Formatting** and other Word toolbars by choosing **Toolbars** from the **View** menu.

For any model document or active, compiled section that is currently open in the Editor window, you can see:

- its content (rules, text, placeholders).
- its properties (word processor type, and so on) in the **Properties** dialog box, accessed from the **File** > **Section Properties** menu.
- if the **Outline** window is visible, an outline or tree view of the model document, starting with the master section, once its sections are successfully compiled.
- additional sub-sections that you can open in multiple **Editor** windows (Window menu).
- *Toolbars* on page 28

The Outline Window

Also called the tree view, the **Outline** window displays the hierarchy of the model document and its included sections. It is updated when you compile a model document or sub-section.

In the tree view, sections in the model document are labeled with the following icons:

-  indicates no errors or failures
-  indicates a compilation error or failure
-  indicates if the section has been accessed and possibly changed but not compiled
-  an arrow indicates a circular reference among sections

If you added sections to the model document, but have not compiled it, you will not see these sections in the tree view. You must first refresh the tree view.

View the sections in the model document by clicking on the branches of the tree view. You can click “+” or “-” symbols to expand or collapse the branches.

Refreshing the Tree View

The tree view is refreshed or updated when you compile or generate the model document. This occurs when you select any of the following options in the **Author** menu: **Compile Section, Compile All Changed, Rebuild Model Document**, or **Generate**.

If you open a .CMS created in an earlier version of Author, clicking anywhere within the tree view will automatically convert the file to the new master section format and save it.

The Compile/Generation Log Window

The Compile/Generation Log window displays the compile/generation log. Each time you compile a section, the log is overwritten. To view the log, select **View** > **Compile/Generation Log** from the **Author** menu or click the **Toggle Compile/ Generation Log** button on the toolbar. Save the log if you need to have a record of a particular compilation.

Toolbars

When an active **Editor** window is open, you can see the toolbars that you set up in Word and a toolbar for IStream Author along the top of your screen.

Show or hide IStream Author toolbars using the **Author** > **View** menu. Show or hide **Formatting** and other Word toolbars by clicking **View** > **Toolbar**.

To add, delete, or change buttons on the toolbars provided with Word, you must exit IStream Author, open Word, then customize the toolbars using Word's **Tools** > **Customize** command as described in *Adding Page Setup to the IStream Author Toolbar* on page 23.

The toolbar buttons that are available on the Word toolbar (**New**, **Open**, **Save As,** and **Print Preview**) do not function in IStream Author. Instead, use IStream Author's toolbar buttons and menu items to access these functions. See also *The IStream Author Interface* on page 22.

IStream Author Options

Use the IStream Author **Options** dialog box to select the default repository. You need to select a repository to store your model documents and model sections.

Method: Display the IStream Author options

- From the **Author** menu, select **Options**.

  The **Options** dialog box displays.

See *Setting the Default Repository* on page 154 for instructions on setting the default repository.

See *Setting the Word Processor to Print the Log* on page 30 for instructions on setting Word to print the compile/generation log.

Setting the Word Processor to Print the Log

Note: If you do not set a word processor to print the log, the log will be sent to your default printer.

1. On the **Options** dialog box, select the **Generation** tab.
2. Click **Browse** to search for and select a word processor or text editor.

   The **Select Word Processor** dialog box displays.
3. Search for and select a word processor or text editor program.
4. Click **Open** to select the program and close the **Select Word Processor** dialog box.
5. Click **OK** to close the **Options** dialog box.
6. Click **File** > **Print Compile/Generation Log**.

   The log is printed.

Editing Default Values

This sections explains:

- *Editing PDF System Defaults* on page 31
- *Editing Charts and Graphs Defaults* on page 32

Editing PDF System Defaults

You can set a default for all model documents by specifying if you want to enable the ability to export to PDFs using hyperlinks and bookmarks, and by specifying which Word styles you want to use to create bookmarks. Note that choosing to enable exporting to PDFs using hyperlinks and bookmarks will increase the time it takes to generate the PDF.

You can also define hyperlinks and bookmark settings for individual model documents, which will override any default for all model documents you have set. See *Defining PDF Properties for a Model Document* on page 172.

Important: The template attached to a master section in a model document overrides any templates attached to sub-sections of the model document. If sub-sections are using a different template than the master section, the template for the master section is what will be used when the model document is generated. Bookmarks will be created according to the styles defined in the template attached to the master section.

For more information about bookmarks and hyperlinks, see

- *Hyperlinks and PDF Bookmarks* on page 171
- *Hyperlink and Bookmark Considerations* on page 171

Method: Edit PDF system default properties

---

1. On the **Author** menu, click **Edit PDF Defaults**.

   The **PDF Default Properties** dialog box is displayed, with the current default template shown in the **Defined Templates** drop-down list and selected default styles shown in the **Include Style** list. Available styles appear in the **Don't Include** list. The default displayed in the **Render to PDF** drop-down list is **Without Preserving Hyperlinks and Creating PDF Bookmarks**.

2. Select an option from the **Render to PDF** drop-down list.

3. Do one of the following steps:

   - in the **Defined Templates** drop-down list, select a Word template stored on your local computer (templates that have previously defined PDF bookmark styles and levels), or accept the default template shown
   - click **Browse** to navigate for alternate Word templates that you want to define default PDF bookmark styles for

Note: When you select a different template, the styles shown in the **Include Style** and **Don't Include** lists refresh to reflect the styles previously selected for that template. If no styles were previously selected for that template, the default styles common to all Word templates (Heading 1, Heading 2, and Heading 3) are shown in the **Include Style** list, along with available styles that can be selected in the **Don't Include** list.

4. Click a style in the **Don't Include** list to highlight it and then click <-- to add it to the **Include Style** list.

Tip: To add multiple styles to the **Include Style** list, use the **Ctrl** or **Shift** key while clicking on one or more styles.

Note: You cannot add more than 100 styles to the **Include Style** list.

5. Click a style in the **Include Style** list to highlight it and then click --> to add it to the **Don't Include** list.

6. Click **Clear Template Defaults** to remove all the styles in the **Include Style** list, except for the Word template default styles: Heading 1, Heading 2, and Heading 3.

Note: If you click **Clear Template Defaults**, do not add any styles, and then click **OK**, the template you are currently editing no longer appears in the **Defined Templates** list. An exception to this is if you are editing `Normal.dot` (the Word default template), which continues to be shown in the **Defined Templates** list, but with all the styles removed from the **Include Style** list.

7. Click **Apply** to save the style selections for the template you are currently editing, allowing you to continue editing other templates.

8. Click **OK** to complete editing your templates and return to IStream Author, or **Cancel** to omit any changes.

Editing Charts and Graphs Defaults

This section describes how to edit the defaults for charts and graphs for an individual IStream Author user. These defaults will be displayed when the IStream Author user inserts a chart or graph (see *Inserting Charts and Graphs* on page 213).

Note: System defaults for charts and graphs are set when IStream Author is installed and cannot be changed. When you follow the procedure in this section to edit IStream charts and graphs defaults, you are configuring the defaults for the current user. Other IStream Author users will have to set their own defaults.

Warning: Any titles you enter on the **Chart Area**, **X Axis**, or **Y Axis** tabs might not completely display when the chart or graph is generated, depending on the font size you use, the title's length, or the chart or graph's size.

Method: Edit the default values for charts and graphs

1. Choose **Edit Chart Defaults** on the **Author** menu and select one of the following items from the menu:

   - **General Properties** – this opens the **General Chart Properties** dialog box where you select default values for:

     - **By Column** or **By Row**
     - **Chart Type**

     These defaults appear on the **General** tab of the **Chart Properties** when you insert a chart or graph.

     (The **Image Format** field is read only.)

   The other choices on the **Edit Chart Defaults** menu allow you to select the default values for various types of charts and graphs:

   - **Pie 2-D** – two-dimensional pie chart default settings
   - **Pie 3-D** – three-dimensional pie chart default settings
   - **Column 2-D** – two-dimensional column chart default settings
   - **Column 3-D** – three-dimensional column chart default settings
   - **Bar 2-D** – two-dimensional bar chart default settings
   - **Bar 3-D** – three-dimensional bar chart default settings
   - **Line** – line graph default settings

   When you select one of these chart or types, a dialog box opens for that type. On this dialog box, you enter the default values for this type of chart or graph. These defaults will then appear on the **Chart** dialog box when you insert this type of chart or graph.

   Some charts include a **Ignore SETFORMAT** and **Circular Layout** check boxes on the **Legend and Data Labels** tab. For information about these fields, see *Using Circular Layout with Pie Charts* on page 216 and *Using SETFORMAT with Charts and Graphs* on page 216.

2. While making any changes:

   - click **Apply** to immediately change the default settings and keep the dialog box you are working in open so you can further modify settings on other tabs
   - click **OK** to update all your changes and close the dialog box
   - click **Cancel** to close the dialog box and discard any changes you have made since opening this dialog box, or since you clicked **Apply**

Using EasyText

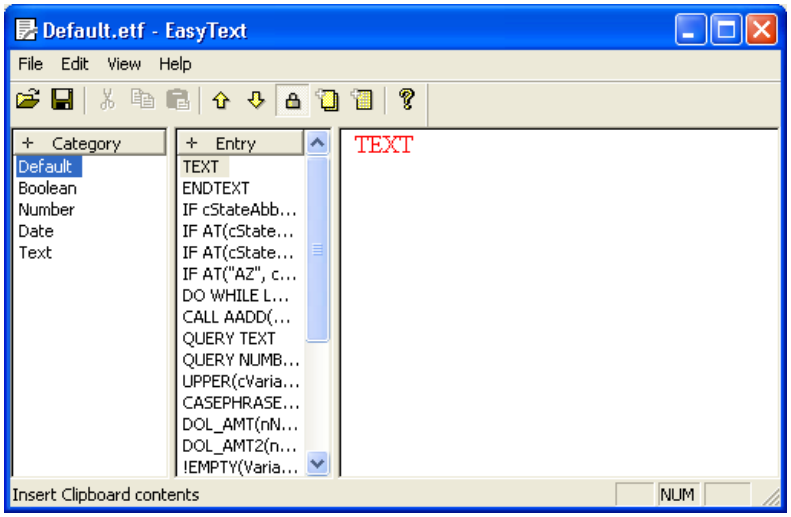
The **EasyText** feature in IStream Author allows you to easily drag and drop a formatted text item into a model document.

Method: Use the EasyText feature

- From the **Author** menu, select **EasyText**, or select the EasyText button from the toolbar:

  The EasyText clipboard opens.



The EasyText clipboard is organized into three columns:

- **Category** – contains groupings of the EasyText entries
- **Entry** – the names of the individual EasyText items
- the text panel that contains the actual text of the EasyText entry

From the EasyText clipboard, you can perform the following tasks:

Method: Insert an EasyText entry into your model document

- Click and drag an **Entry** name into your model document.

Method: Create a new EasyText entry

Select the **Category** where you want the EasyText entry to be added, then complete one of the following steps:

- Right-click within the **Entry** column and select **New Entry**.
- Click the new entry button. 
- Highlight a section of text within the model document, then drag it into the **Entry** column within the EasyText pane.
    - The text panel displays the contents of the entry that you have just dragged into EasyText. If the **Auto_name** feature is enabled for the category of the added EasyText entry, then the entry is automatically named based on the text of the entry. If not, the entry is called **New Entry** by default. Either way, you can rename an entry by right-clicking it and choosing **Rename**.

      For instructions on how enable and disable the Auto_name feature for a category, see *The EasyText Auto_name Option* on page 36.

Method: Create a new EasyText category

Complete one of the following steps:

- Right-click within the **Category** column and select **New Category**.
- Click the new category button. 

Method: Edit an EasyText entry

1. Click the **Entry** name.
2. Click the padlock button to unlock the entry.
3. Edit the text as necessary.
4. Click the padlock button again to save your changes and relock the entry.

Method: Rename an EasyText entry

1. Right-click the **Entry** name and select **Rename**.
2. Enter the new name for the **Entry**.

Method: Sort the categories or entries

Click the **Category** or **Entry** column headers. The sort order will cycle through descending, ascending and the original order that the items were created.

Note: EasyText may not retain all Word formats and styles.

The EasyText Auto_name Option

For each EasyText category, you can configure whether entries created by dragging text from a model document are automatically named.

If the **Auto_name** feature is enabled, an entry created by dragging in text from a model document is automatically named to include text from the body of that entry. For example, if you select and drag the following into EasyText to create a new entry:

State = "MN" .OR. "IL"

the name that appears in the **Entry** column is **State = "MN" .OR. "IL"**, when the **Auto_name** feature is enabled for the corresponding EasyText category.

When **Auto_name** is disabled, an entry that you create by selecting and dragging model document content is named **New Entry** by default.

You can right-click an entry to rename it at any time, regardless of whether or not the **Auto_name** feature is enabled.

Method: Enable or disable the Auto_Name option

1. Right-click a **Category**.
2. You will see the **Auto_name** option on the right-click menu. A check mark next to **Auto_name** means that the option is currently enabled.
   - If there is a check mark next to **Auto_name**, click **Auto_name** to disable the option.
   - If there is no check mark, click **Auto_name** to enable the option.

Chapter 3

Authoring Reference

This chapter provides a comprehensive explanation of the Authoring language and how to use it to author model document sections for generation.

This chapter describes:

The Authoring Language

The topics in this section give a brief description of the elements used in the authoring language.

Rules and Functions

A rule is a command that instructs Assembler to insert text in a generated document in a particular way. A function can be a single reference, or a combination of references, variables, and functions that transforms data, resulting in a return value (for example, a calculated value).

Boolean Logic

In boolean logic, all values are reduced to TRUE or FALSE. This logic forms the basis for the rules, expressions, and functions in IStream Author that are used to retrieve data from data sources. For more information about this topic, and some algebraic examples, see *Using Boolean Logic* on page 50.

Rule Requirements

When entering rules in your sections, you can use upper or lower-case letters, however you should be consistent. If you use the Rule Wizard in IStream Author to insert a rule, uppercase letters are automatically used. This makes the rule easier to find in a block of text when you check your section.

All rules must be followed by a hard return (represented by a ¶ character in Word). You can use Microsoft Word to display hard returns from the **Tools > Options** > **View** menu.

All parameters and arguments in a rule must be separated by one or more spaces. For more information see *Authoring Standards* on page 49.

InfoSources

The IStream Document Manager environment can be quite complex. There are many elements that are used to produce and store documents. InfoSources are used in IStream Author to communicate with these resources.

InfoSources are like bridges that IStream Author uses. InfoSources point to either:

- the database that contains the data for the output document
- document storage locations (repositories)

The IStream Document Manager environment needs to be planned carefully to give users access to the required resources.

The IStream Document Manager administrator plays a key role in planning and implementing InfoSources. Authors need to understand InfoSources and how they work because InfoSources may need to be created on each individual IStream Author workstation.

Types of InfoSources

IStream Document Manager uses the following InfoSource types:

- IStreamDM
- File system
- DynaFS
- ODBC database
- User database
- IStreamXML

Accessing InfoSources

Through IStream Author, a user uses File System and IStream Document Manager InfoSources to open model documents or any of their sections.

Through the Reference Wizard, users navigate within the InfoSource to create references to particular items in the corresponding information sources.

For UserDB, ODBC and IStreamXML types of InfoSources, users do not access the data itself, but data definitions, and create references to them.

Local.idb

All information about your InfoSources is stored in the `local.idb` file. By default, this file is located in the `Shared` directory where IStream Author is installed.

Note: If IStream Author is reinstalled, the `local.idb` is overwritten. You therefore should make a copy before reinstalling IStream Author.

To generate your documents both locally and remotely, matching InfoSources must be set up on both your local computers and on the server version of IStream Assembler.

Repository Type InfoSources

This section describes InfoSources where documents are stored.

IStreamDM

In this type of InfoSource, model documents are stored in the IStream Document Manager.

File System

In this type of InfoSource, a document (a model document or IStream document) repository is placed in a different location from IStream Document Manager. This can be a file system on a local computer, or on a shared network drive.

DynaFS

DynaFS is a special system InfoSource, used during document generation. As a system InfoSource, it cannot be modified or deleted. DynaFS should not be used during document coding or creation.

Data Type InfoSources

This section describes InfoSources used for retrieving data.

ODBC Database

This type of InfoSource points to an Open Database Connectivity (ODBC)-compliant database that contains the data for the model document. This InfoSource is used when an author is using the Query coding method in the model document to access data.

User Database

This type of InfoSource points to the Open Database Connectivity (ODBC)-compliant database that contains the data for the model document. This InfoSource is used when an author is using a mapping in the InfoSource to access data.

IStreamXML InfoSources

This type of InfoSource provides access to data that is located in an XML file.

IStream InfoConnector

IStream InfoConnector is the application used to add, delete and configure InfoSources.

Accessing IStream InfoConnector

You open the InfoConnector using the **Start** menu.

Method: Open the InfoConnector

- Click **Start** > **Programs > IStream > InfoConnector**

  IStream InfoConnector opens.

Note: If you find that IStream Author is not working properly, it could be because standard InfoSources have not been configured accurately. Consult with your IStream Author Administrator.

For more information about how to use the IStream InfoConnector, see the *IStream InfoConnector Online Help*.

FileSystem InfoSources

There are two categories of InfoSources – document repository and data storage.

The FileSystem InfoSource points directly to a storage area on your network. It is a repository for model documents or IStream documents located somewhere other than on IStream Document Manager.

Note: For IStream Author to recognize any new FileSystem or IStream Document Manager InfoSource, IStream Author must be closed when you create the InfoSource.

Method: Add a FileSystem InfoSource

1. Select **Add** from the **InfoSource** menu or click the **Add New InfoSource** button.

   The **InfoSource Type List** dialog box appears.
2. Select the type of InfoSource to add, in this case, the **File System** InfoSource.
3. Select **FileSystem**, then click **OK**.

   The **Configuration for File System type InfoSource** dialog box appears.
4. Give the InfoSource a unique name.

Important: Do not leave any spaces or insert special characters in the InfoSource name because this will cause generation errors.

5. Enter a description of your InfoSource.
6. In the **Location** field, set the location to the physical location on the local computer or the network drive where the model documents will be stored.
7. Click **OK**.

The new InfoSource will be shown in the **InfoSource** list window.

UserDatabase InfoSources

The UserDatabase InfoSource points to a database through a datasource that is created in ODBC Administrator.

A UserDatabase InfoSource accesses data using mappings that define a set of hard-coded SQL queries to the user database. Access to user databases and lookup tables is provided through ODBC drivers.

Method: Add a UserDatabase InfoSource

1. Open the InfoConnector through the **Start > Programs > IStream** menu.
2. Click **Add**.
3. Select the **UserDatabase InfoSource** type, then click **OK**.
4. Enter the name of the InfoSource.

Important: Do not leave any spaces or insert special characters in the InfoSource name because this will cause generation errors.

5. In the **Datasource** field, select the appropriate datasource for this InfoSource.
6. Click **OK**.

Using InfoSources in Model Documents

InfoSources do not work in isolation. They give directions in the model document coding. Data type InfoSources indicate in a model document the location of the database information. A Repository type InfoSource indicates in a model document where to store or retrieve the model document section files.

There are many IStream Author rules that use the InfoSource references.

When a model document section needs to retrieve data from a database, a database InfoSource is coded into the model document. Each time a model document section requires the location of a section, a repository InfoSource is coded into the model document.

Using Operators

An operator is a symbol that performs an action within an expression. Operators allow you to manipulate numbers and text. For example, boolean operators allow you to test whether a condition is true or false. Relational operators allow you to compare one value to another. (For the basics of Boolean Logic, see *Using Boolean Logic* on page 50.)

Boolean operators are often used with IF... ELSE... ENDIF, IF... ELSEIF... ELSE... ENDIF and IF... ENDIF rules to establish criteria for the values retrieved from the database.

You can also use these boolean operators in combination to further refine an expression. For example, if you need to ensure that a value meets two criteria, and that it is greater than or equal to another criterion you specify, you can refine that particular value by using operators in combination.

The following section provides more information about using operators in IStream Author.

Operator Types

An operator is a symbol that represents a specific action. An operand is an object that is manipulated by that action. For example, in the expression 5 + x, x and 5 are operands, and + is an operator. All expressions have at least one operand.

Note: Logical operators do not function the same as mathematical operators. An equals sign, for example, indicates True logically, although mathematically it would indicate “the same as”.

= Equal

With a character field, this operator does not display an exact match. For example, `box=boxes` would be True because all three characters on the left match the first three characters on the right. The other characters are ignored. If you want an exact match, use == Exactly equal.

== Exactly equal

Use this operator to compare two values or to indicate the values must be exactly equal. With this operator, box==boxes would be False.

!= Not equal

This operator is a combination of !(Not) and =(Equal). Use it to indicate not equal. This operator returns all of the values that are not equal to the operand in the expression. It is used when you want to eliminate a particular value from consideration while accepting all other values.

< Less than

Use this operator to indicate less than.

<= Less than or equal to

Use this operator to indicate less than or equal to.

> Greater than

Use this operator to indicate greater than.

>= Greater than or equal to

Use this operator to indicate greater than or equal to.

+ Addition or Concatenation

Use this operator to indicate the addition of numbers or concatenation of characters. For details, see *Concatenating Values* on page 45.

- Subtraction

Use this operator to indicate subtraction of numbers (78-50=28). Trailing spaces from the first parameter are removed.

* Multiplication

Use this operator to indicate multiplication.

/ Division

Use this operator to indicate division.

.AND. Boolean and

This operator evaluates two or more conditions or operands together in one expression. Both conditions must be true to meet the criteria of the expression. It is used in search engines to restrict search criteria to certain values.

The found items must match all of the criteria given to be returned as a result in the search. Although the format is slightly different than that used in a section, your word processor uses a Boolean AND to help you find particular files.

.OR. Boolean or

This operator evaluates two or more conditions, and, if either is true, then it meets the criteria of the expression. This boolean operator is often used in search engines to narrow search criteria to certain values.

True/False

.T.
.F.

Use these logical operators to establish whether a condition must be true or false.

! Not

The ! operator indicates NOT. In IStream Author, you can use ! with the = sign (!=) to indicate NOT EQUAL. You can also use ! with EMPTY (!EMPTY) to

indicated NOT EMPTY. For more information, see *!= Not equal* on page 43 *EMPTY and !EMPTY* on page 107.

Concatenating Values

You can join or *concatenate* text strings and add numeric values by defining the values and then using the *plus* symbol (+). The following sections describe how to concatenate various combinations of values.

Concatenating Strings

You can assign characters to variables and then concatenate them:

Example:
```
DEFINE STRING_A "ABC"
DEFINE STRING_B "XYZ"
DEFINE STRING_C STRING_A + STRING_B
```

`STRING_C` would be rendered as `ABCXYZ`.

Adding Numbers

You can assign numeric values to variables and then add them:

Example:
```
DEFINE VALUE_A 5
DEFINE VALUE_B 10
DEFINE SUM VALUE_A + VALUE_B
```

`SUM` would be rendered as `15`.

Concatenating Strings with Numbers

To concatenate a string value with a number, you first need to convert the number to a string using the `str` function:

Example:
```
DEFINE VALUE_A 123
DEFINE STRING_A "ABC"
DEFINE RESULT_A str(VALUE_A) + STRING_A
```

`RESULT_A` would be rendered as `123ABC`.

Concatenating Numbers as Strings

You can concatenate two numeric values as strings. Note that the final value is *not* the *mathematical* total of the two strings, but the *concatenated* value.

Example:
```
DEFINE Value_A 123
DEFINE Value_B 456
DEFINE Result_A str(Value_A) + str(Value_B)
```

`Result_A` would be rendered as `123456`.

Rules and Functions Overview

This section outlines the rules and functions that are available for coding model documents in IStream Author. Each rule or function has a description of what it is and how to use it, along with a few examples.

This section describes:

- *Rules and Functions Table* on page 46
- *Authoring Standards* on page 49
- *Rules* on page 52
- *Functions* on page 89

Rules and Functions Table

The following table can help you determine which rules or functions you might need for certain authoring tasks.

Topic	Rule(s)/Function(s)
Array functions	• *AADD* on page 90 • *ASORT* on page 92
Call functions for integration with external programs	• *CALL APP* on page 95 • *DLL* on page 103
Data Rules/Functions (InfoSource use, looping, querying)	• *BREAK on page 53* • *DO WHILE on page 60* • *IF* on page 64 • *ELSEIF on page 61* • *ENDIF on page 62* • *FORCE NEXT on page 63* • *INITIALIZE on page 73* • *NEXT on page 75* • *QUERY on page 75* • *SELECT on page 79* • *SELECT ALL on page 80* • *UNUSEIS on page 87* • *USEIS on page 87*

Topic	Rule(s)/Function(s)
Date and Time functions	• *CDOW* on page 97 • *CMONTH* on page 100 • *CTOD* on page 101 • *DATE* on page 101 • *DAY* on page 102 • *DMY* on page 104 • *DOW* on page 106 • *DTOC* on page 106 • *DTOS* on page 107 • *GETDAY* on page 110 • *GETMONTH* on page 113 • *GETYEAR* on page 114 • *MDY* on page 124 • *MONTH* on page 125 • *YEAR* on page 146 • *YMD* on page 146
Debugging functions	• *DISPVAR* on page 102 • *DOCERR* on page 104 • *SHOWRULES* on page 139 • *ENDJOB* on page 62
Define a variable	• *DEFINE* on page 54
Financial functions	• *DOL_AMT* on page 105 • *DOL_AMT2* on page 105 • *NUMBER* on page 126 • *NUMBER2* on page 126 • *SPELLNUMBER* on page 140
Include another section	• *INCLUDE* on page 67
Information functions	• *EMPTY and !EMPTY* on page 107 • *ISALPHA* on page 116 • *ISDIGIT* on page 116
Locale setting functions	• *SETFORMAT* on page 136 • *SETLANGUAGE* on page 139

Topic	Rule(s)/Function(s)
Lookup and reference functions	• *The waiting period is 1 month.* on page 98 • *FIELD* on page 108 • *FINDFILE* on page 109 • *GETFILE* on page 110 • *LOOKUP* on page 121 • *MDY* on page 124 • *MYUSERID* on page 126 • *PROMPT* on page 131 (LOCATEFOUND, LOCATEITEM and LOCATENEXT are not supported)
Mathematical functions	• *ABS* on page 91 • *EXP* on page 108 • *INT* on page 115 • *LOG* on page 120 • *MAX* on page 123 • *MIN* on page 124 • *POWER* on page 130 • *ROUND* on page 135 • *SQRT* on page 141 • *TRUNCATE* on page 144
Picture Functions	• *GETIMAGE* on page 111 • *ISIMAGE* on page 117 • *SETIMAGEFORMAT* on page 137

Topic	Rule(s)/Function(s)
Text rules and functions: • define a text block • concatenate (join) • manipulate text	• *TEXT* on page 81 • *TEXT {}* on page 85 • *TEXT DEFAULT {}* on page 85 • *ENDTEXT* on page 62 • *ENDTEXT+* on page 62 • *ALLTRIM* on page 91 • *AT* on page 93 • *CASEPHRASE* on page 96 • *CASEWORD* on page 96 • *CHECK_DAY* on page 97 • *CHECK_MTH* on page 98 • *ISLOWER* on page 118 • *ISUPPER* on page 119 • *LEFT* on page 119 • *LEN* on page 120 • *LOWER* on page 122 • *LOWTRIM* on page 122 • *LTRIM* on page 123 • *MYSTR1* on page 125 • *PADC* on page 127 • *PADL* on page 128 • *PADR* on page 129 • *REPLICATE* on page 134 • *RIGHT* on page 134 • *RTRIM* on page 136 • *SPACE* on page 140 • *STR* on page 142 • *SUBSTR* on page 142 • *TRIM* on page 143 • *UPPER* on page 144 • *UPTRIM* on page 145 • *VAL* on page 145

Authoring Standards

Note the following authoring standards when creating rules and functions:

- a rule must be on a line by itself
- a rule must be followed by a hard return or paragraph marker (¶); a hard return symbol is inserted when you press **Enter** or **Return**
- if a rule statement is too long to fit on one line, do not break the rule statement with a hard return; let it wrap instead
- character information must be surrounded by double straight quotes " "
- numeric values do not have quotes
- logic constants must be surrounded by periods (.T. or .F.)

- all parameters and arguments in a rule statement must be separated by at least one space
- data can be entered in either upper or lower-case, but should be consistent
- every TEXT has an ENDTEXT, every IF has an ENDIF, and so on
- the supported data types are: logical, date, numeric, and string

Reserved Words in Field and Table Names

You cannot use any of the following as field or table names:

- IStream rules or functions
- IStream operators
- SQL commands

Related Topic

- *Using Boolean Logic* on page 50

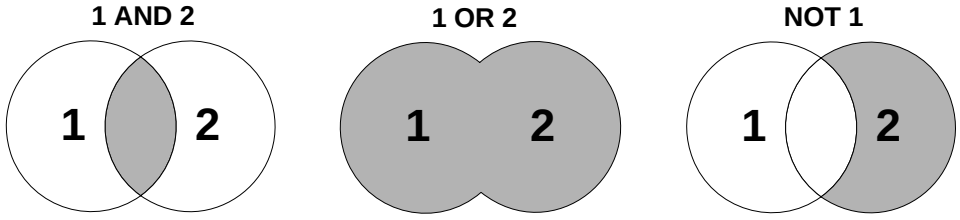
Using Boolean Logic

Boolean logic is a form of algebra where all values are reduced to either True (yes/on) or False (no/off). In the binary numbering system, true is equivalent to 1, and false is equivalent to 0. There are no intermediate values. When combined with logical operators, such as + (or) or ! (not), you can form expressions which manipulate the handling of data.

Note: When using logical persistent variables, you must use .T. or .F. to initialize the variable. Do not use “True” or “False”.

Search engines use Boolean logic to evaluate search criteria and find values which match the required result of the Boolean expression. Using the word AND narrows the results obtained in a search, while using the word OR broadens the results. NOT eliminates certain results.

Graphically, the three primary operators of AND, OR, and NOT look like this:



The shaded area represents the portion of the circles that meet the given criteria.

- In the AND condition, the shaded area must be part of both circles for it to resolve as true.
- In the OR condition, the shaded area represents both circles because OR indicates that both values are true.

- In the NOT condition, the shaded area represents the area of the second circle that is NOT in the first circle.

In an insurance context, for example, the variables sought may be <live in California> OR <Over 25 years of age> AND <Employed for more than three months> NOT <male>. The results would be any female living in California who has been employed for more than three months, and is over 25 years of age.

Combining operators and functions in expressions enables you to specifically define the data that is pulled into your assembled document.

IStream Assembler uses this logic, and other logical expressions coded into model documents, to determine what values to retrieve from a database. Expressions, or queries, are often classified by the type of value they represent.

Types of expressions include:

- *Boolean expressions*, which evaluate to either TRUE or FALSE.
- *integer expressions*, which evaluate to whole numbers such as 3 or 100.
- *floating-point expressions*, which evaluate to real numbers such as 3.421 or -0.005.
- *string expressions*, which evaluate to character strings.

Example: Following is an example of a Boolean expression:

```
IF p_female = .T. .AND. p_age > 65
TEXT
This applies to all females over the age of 65.
ENDTEXT
ENDIF
```

The statement “This applies to all females over the age of 65.” is printed if p_female is true and p_age is greater than 65.

To use Boolean logic in IStream Author, you should make sure that you understand a few important operators, such as .AND. , .OR. , .T. , and .F. . For more information on these operators, see *Operator Types* on page 43.

Related Topic

- *Reserved Words in Field and Table Names* on page 50

Rules

The following rules can be used in any section of your model document, however some rules must be used together: see *Paired Rules* on page 52. Examples show how a rule is used to achieve the appropriate result.

BREAK	ENDJOB	QUERY
CALL	ENDTEXT	SELECT
COMMENTS	ENDTEXT+	SELECT ALL
DEFINE	FORCE NEXT	TEXT
DO WHILE	GENERATE	TEXT {}
ELSE	IF	TEXT DEFAULT {}
ELSEIF	INCLUDE	TEXT TAG
ENDDO	INITIALIZE	UNUSEIS
ENDIF	NEXT	USEIS

Paired Rules

The following rules must be paired together when used.

TEXT	• ENDTEXT, • ENDTEXT+ (where ENDTEXT+ is a modification of ENDTEXT)
QUERY	• NEXT • BREAK and NEXT (where BREAK is optional, NEXT is always mandatory)
DO WHILE	ENDDO
USEIS	UNUSEIS
INITIALIZE	• NEXT, • BREAK with NEXT • FORCE NEXT with NEXT (where BREAK or FORCE NEXT is optional, NEXT is always mandatory)
IF	The syntax of the IF structure is: • IF (mandatory) • followed by zero or more: ELSEIF • followed by zero or one: ELSE • followed by a mandatory: ENDIF

BREAK

This rule breaks loops (it ensures the rule is executed only once) in the following rules:

- QUERY / NEXT
- INITIALIZE / NEXT
- DO WHILE / ENDDO

Type this rule in a separate line within the body of the loop. When this rule is encountered, control is passed to the next command after the loop closing rule (in effect, NEXT or ENDDO) nearest to the BREAK rule.

For an example, see *QUERY on page 75*.

CALL

Syntax: CALL functionname

Use the CALL rule to execute these functions:

- APP
- AADD
- DISPVAR
- DOCERR
- SETFORMAT
- SETLANGUAGE
- SHOWRULES

For more information on functions, see *Functions* on page 89.

Example:
```
CALL SETFORMAT ("FRENCH")
***refers to the French formats in the Formats
***file.
TEXT
<(MDY(CTOD("02/12/2007")))>
ENDTEXT
CALL SETFORMAT ("ENGLISH")
***resets the formats file back to the English
***default.
```

Generated result: fev 12, 2007

COMMENTS

Syntax: *** text of comment

Because coding can be complicated, it is useful to record comments in your section to help you remember important information. This also assists other authors who may be working on or reviewing this section. Comments do not interfere with processing and do not appear in the assembled document.

Place comments on a separate line. If the comment is too long to fit on one line, break it with a hard return (¶). The continuation of the comment on the next line must be preceded by an asterisk (*).

Make sure that the comment is not placed between TEXT and ENDTEXT or it will appear as printed text.

Important: Always include a hard return (¶) at the end of a comment that is inside a table cell.

Tip: Because nested IF statements can become confusing, you may want to keep track of the openings and closings of IF statements nested within other IF statements by appending comments to the ENDIF portion of the rule:

```
IF c_life = .T.
*** check to see if there is life coverage.
IF !EMPTY(c_flatamt)
*** check to see if there is a lifetime maximum.
TEXT
Lifetime maximum <dol_amt(c_flatamt)>
ENDTEXT
ELSE
TEXT
Lifetime maximum is 50000.
ENDTEXT
ENDIF
*** matches if there is a lifetime maximum.
ELSE
TEXT
Text class has no Life coverage.
ENDTEXT
ENDIF
*** matches if there is life coverage.
```

DEFINE

Syntax: DEFINE variableName variableValue

The DEFINE statement is used to create variables to hold frequently used information, and also to create variables that are not in the database (for example, counters). The values of a defined variable are held in memory for the full generation of a document. Once a variable is defined, you may change its value or reuse it by issuing another DEFINE statement for that variable.

IStream Document Manager can therefore generate documents more quickly with variables than if the same information was accessed through multiple INITIALIZE... NEXT or QUERY....NEXT loops which access the database.

Variables created by DEFINE must have a unique name within the model document. In effect, you cannot redefine variables from the database using the same name as the variable from the database. A character or string constant `variableValue` is enclosed with quotation marks when being assigned in a `variableName`.

When creating a `variableName`, you usually need to know what type of data – string, numeric, logical, or date – will be contained in the `VariableValue` so that this data can be properly used. The following variable naming conventions are recommended:

Date Type	Prefix	Variable Name Example	DEFINE Example
Character	c	cInsuredName	DEFINE cInsuredName ""
Numeric	n	nPremiumAmount	DEFINE nPremiumAmount 0
Boolean / Logical	b or l	bWholeLifeIndicator or lWholeLifeIndicator	DEFINE bWholeLifeIndicator .T.
Date	d	dEffectiveDate	DEFINE dEffectiveDate CTOD("04/03/2007")
Array	a	aClassDescription	DEFINE aClassDescription {}

Example:
```
DEFINE cPolicyNo "Life0012"
*** defines a string variable
DEFINE nAmtInsured 75000
*** defines a numeric variable
DEFINE cInsuredName InsName
    ***defines a string variable where the value is obtained
    ***by referring to another variable. INSNAME in this
    ***example could be a field in the Database table, "John
    ***Doe" is the value of the field
TEXT
Insured Name: <cInsuredName>
Policy Number: <cPolicyNo>
Amount Insured: $<nAmtInsured>
ENDTEXT
```

Generated result:

Insured Name: John Doe

Policy Number: Life0012

Amount Insured: $75000

You can use Universal Information Source Reference (UISR) syntax for both the `variableName` and `variableValue` arguments in the `DEFINE` rule.

Using UISR Syntax with DEFINE

You can use Universal Information Source Reference (UISR) syntax for both the `variableName` and `variableValue` arguments in the `DEFINE` rule.

The syntax for using UISRs with the DEFINE rule is:

```
DEFINE ISC:variableNameReference ISC:variableValueReference
```

where:

- `ISC` is the name of an InfoSource. This can only be a valid InfoSource name or a string constant.
- `variableNameReference` can be a combined variable name, a string constant, or a string expression. If it is a variable, it must already be defined. If it is a database field, a rowset must be queried and available.
- `variableValueReference` can be a combined variable name, a string constant, or a string expression. If it is a variable, it must already be defined. If it is a database field, a row must be queried and available.

Using the RecordsetName Identifier

Where different queries are made against the same InfoSource or a table in an InfoSource, you can reduce ambiguity between the multiple queries with a `RecordsetName` identifier. The `RecordsetName` identifier can be any valid name.

```
DEFINE ISC.RecordsetName:variableNameReference
ISC.RecordsetName:variableValueReference
```

Omitting the InfoSource Name

If the InfoSource name is omitted in the UISR for the `variableNameReference`, the second colon (:) must be included to distinguish it from a simple `DEFINE` rule with a hard-coded variable name:

```
DEFINE :variableNameReference :variableValueReference
```

If the InfoSource name is omitted in the UISR for the variableValue, the second colon can be omitted, as here:

```
DEFINE :variableNameReference variableValueReference
```

Using the DEFINE Rule

The `DEFINE` rule cannot add or remove InfoSource names from the model document. The following rules and functions *can* add InfoSource names:

- QUERY
- USEIS
- INITIALIZE

Using Rules and Functions

The following rules and functions can remove InfoSource names:

- UNUSEIS

- NEXT
- BREAK

Examples

Example 1 – Querying a Table

This example shows how a query can be used to obtain information from a table where both the variable name and value are variables.

In this example, the InfoSource is called `DefineTest`.

```
*** This code reads the values from the Plan_Var
*** table and assigns the value to Var_Name.
QUERY "SELECT PLAN_VAR.Var_Value, VAR_TYPE.Var_Name FROM PLAN_VAR,
VAR_TYPE WHERE Policy_Nbr = " + STR(nPolicyNbr) + " AND VAR_TYPE.Var_Typ
= PLAN_VAR.Var_Typ AND VAR_TYPE.Var_ID = PLAN_VAR.Var_ID AND
VAR_TYPE.Level_Cd = PLAN_VAR.Level_Cd","DefineTest", "PlanVar"

*** If variable type is logical, set the variable as .T. (true):
  IF LOWER(SUBSTR(Var_Name,1,1)) = "l"
      DEFINE :Var_Name .T.
  ENDIF

*** If the value is numeric, set Var_Name with the
*** numeric value:
  IF LOWER(SUBSTR(Var_Name,1,1)) = "n"
    DEFINE :Var_Name Var_Value
  ENDIF

*** If the value is a date or character type, set
*** Var_Name with the date or character value:

  IF LOWER(SUBSTR(Var_Name,1,1)) = "c" .OR.
  LOWER(SUBSTR(Var_Name,1,1)) = "d"
    DEFINE :Var_Name Var_Value
  ENDIF
NEXT
```

Example 2 – Using Variations of DEFINE Statement

In this example, the following variations of the `DEFINE` statement can be used. The ISC and the Rowset are optional in the statement and all will give the same result:

```
***Define with ISC in the VariableNameReference
DEFINE DefineTest:Var_Name Var_Value

***Define with Rowset, a period precedes the rowset if ISC is omitted
DEFINE .PlanVar:Var_Name Var_Value

***Define with both ISC and Rowset in the VariableNameReference
DEFINE DefineTest.PlanVar:Var_Name Var_Value

***Define that directly assigns the value of VariableValueReference to
VariableNameReference
```

```
DEFINE :Var_Name :Var_Value

***DEFINE with ISC in both VariableNameReference and
VariableValueReference
DEFINE DefineTest:Var_Name DefineTest:Var_Value

***DEFINE with Rowset in both VariableNameReference and
VariableValueReference
DEFINE .PlanVar:Var_Name alltrim(.PlanVar:Var_Value)

***DEFINE with ISC and Rowset in both VariableNameReference and
VariableValueReference
DEFINE DefineTest.PlanVar:Var_Name DefineTest.PlanVar:Var_Value
```

Using Referenced variableName or variableValue

The `variableName` and `variableValue` arguments can be actual names specified in the define statement, or they can be references to other variables' values.

This can be done by preceding a colon (:) in the `variableName` and/or the `variableValue` parameter to tell the `DEFINE` rule to evaluate the argument. See the following syntax and examples for information.

Note: A `variableName` can contain any type of alphanumeric character. It can start with, contain, or end with a number. However, it must contain at least one character. It can contain an underscore (_), but cannot contain any other special characters. There are no restrictions for what a `variableValue` can contain.

Syntax: DEFINE :variableNameReference :variableValueReference

Note: The colon can be optional on either side of the `variableName` and/or `variableValue` depending on the purpose of the argument.

Example: In this example, `cName` is evaluated to return the value of `cName`. The value of `cName` is "`cPolicy`", which is the name of a variable. The `DEFINE` rule then returns the contents of `cPolicy` as the value for the newly defined `cReference` variable.

```
DEFINE cPolicy "Life0012"
DEFINE cName :"cPolicy"
DEFINE cReference cName
TEXT
Simple variable: <cPolicy>
Another simple variable: <cName>
A referenced variable: <cReference>
ENDTEXT
```

Generated result:

Simple variable: Life0012

Another simple variable: Life0012

A referenced variable: Life0012

Example:
```
DEFINE VarName "dEffDte"
DEFINE dDate "01/04/2007"
DEFINE :VarName dDate
TEXT
Effective Date: <dEffDte>
VarName: <VarName>
ENDTEXT
```

Generated result:

Effective Date: 01/04/2007

VarName: dEffDte

Example:
```
QUERY "Select * from Define","NewDefine"
     DEFINE :var_name :var_value
NEXT

TEXT
<_First_Name>
<1Initial1>
<_Last_Name>
ENDTEXT
```

Generated result:

Fred

!@#$%^&*()

Smith

For more examples of Referenced variableName or variableValue, see *Using UISR Syntax with DEFINE* on page 56.

Persistent Variables

To persist variables, place a dollar sign in front of the variable name. When the IStream document is assembled, the persistent variables are maintained in a list you can view in the document properties. In Assembler, select **File** > **Properties**, and click the **Persistent Variables** tab. You can also see all the defined variables in the **Reference Wizard Local** variables list.

Example:
```
DEFINE $coverage "limited"
DEFINE $deadline "January 1st"
DEFINE $policy "accident"
TEXT
```

Please note that all <$coverage> liability <$policy> policies which have not been renewed in writing by <$deadline> will be cancelled.

```
ENDTEXT
```

Generated result:

Please note that all limited liability accident policies which have not been renewed in writing by January 1st will be cancelled.

EFFECTDATE Variable

You can use the EFFECTDATE variable to set the effective date when generating a document. This date will be used to determine which sections should be included based on the effective and termination dates associated with them.

Example 1

The following example shows how to assign a specific value to EFFECTDATE:

```
DEFINE EFFECTDATE CTOD("01/10/2007")
```

Example 2

The following example shows how to assign a database variable to EFFECTDATE:

```
QUERY "Select startDate from CustTable where CustID='123'",
"CustData"
DEFINE EFFECTDATE startDate
```

Note: If you do not assign a value to EFFECTDATE, then the current system date is used.

DO WHILE

Syntax: DO WHILE varname operator value

It is useful to repeat part of a section using something other than a database for control. For example, use the DO WHILE rule to generate four copies of a cover page.

Complete the rule with ENDDO when you use DO WHILE.

Note: There must be a space between DO and WHILE.

Example:
```
DEFINE count 0
DO WHILE count < 4
INCLUDE contracts:"coverpage"
DEFINE count (count +1)
ENDDO
```

Example: The DO WHILE...ENDDO rule can be used to print information from an array, to have the code be dynamic based on the value of the counter variable.

A sample array would be:

Doe, Jane	Spouse
Doe, Fred	Son
Doe, John	Son
Andrews, Mary	Daughter

```
DEFINE Beneficiaries {}
CALL AADD(Beneficiaries, {"Doe, Jane", "Spouse"})
CALL AADD(Beneficiaries, {"Doe, Fred", "Son"})
CALL AADD(Beneficiaries, {"Doe, John", "Son"})
```

```
CALL AADD(Beneficiaries, {"Andrews, Mary", "Daughter"})
TEXT
```

Beneficiaries

```
ENDTEXT
DEFINE count 1
DO WHILE count <=LEN(Beneficiaries)
TEXT
<Beneficiaries[count,1]>
ENDTEXT
DEFINE count (count + 1)
ENDDO
```

The document would read:

Beneficiaries

Doe, Jane
Doe, Fred
Doe, John
Andrews, Mary

Note: For more information on arrays, see *Using Arrays* on page 147.

ELSE

Syntax: ELSE

Use this rule to dictate what to do when the IF statement evaluates to FALSE.

Use the ELSE rule within the IF...ENDIF rule. An ELSE rule must follow an IF rule.

For an example, see *IF* on page 64.

ELSEIF

Syntax: ELSEIF conditional expression

Use this rule to dictate what to do when the IF statement evaluates to FALSE and you have another condition to evaluate. Use the ELSEIF rule within the IF... ENDIF rule.

For an example, see *IF* on page 64.

ENDDO

Syntax: ENDDO

Use the ENDDO rule to close the DO WHILE rule. For an example, see *DO WHILE on page 60*.

ENDIF

Syntax: ENDIF

Use the ENDIF rule to close the IF rule.

Note: Every IF statement must be paired with an ENDIF statement.

For an example, see *Using the ENDIF rule* on page 66.

ENDJOB

Syntax: ENDJOB

This rule stops generation as soon as it is encountered. It is useful for debugging particular rules, sections or portions of sections.

You must enter rule this on a separate line.

Example: The code below counts from 0 to 49 and stops at 50.

```
DEFINE counter 0
DO WHILE counter <= 100
      IF counter = 50
         ENDJOB
      ENDIF
   TEXT
      <counter>
   ENDTEXT
   DEFINE counter counter + 1
ENDDO
```

ENDTEXT

Syntax: ENDTEXT

Use the ENDTEXT rule to close the TEXT rule. For an example, see *TEXT* on page 81.

Warning: If the ENDTEXT rule is missing after the TEXT rule, your system may stop working while in Word.

ENDTEXT+

Syntax: ENDTEXT+

This rule links a series of text blocks. It must be placed on a separate line. It follows the same semantics as ENDTEXT in that it must match the TEXT rule and close the text block. It also means that the following text blocks, marked with TEXT...ENDTEXT rules or inserted with the help of the INCLUDE rule, are linked with the TEXT...ENDTEXT+ block.

Note: There is no space between ENDTEXT and the plus (+) sign.

This rule replaces <"" (less than, quote quote) in the MOSAIC language, although <"" is still supported.

For an example of how to use ENDTEXT+, see *ENDTEXT* on page 62 and *TEXT* on page 81.

FORCE NEXT

Syntax: FORCE NEXT

Use the FORCE NEXT rule within an INITIALIZE...NEXT loop to instruct Assembler to move to the next record without repeating the processing of rules.

To use FORCE NEXT, you must know the order of the records (index expression) for the table in use, so that you can determine when it is appropriate to skip to the next record. FORCE NEXT is a tool for optimizing your document.

In the following example, the CLASS table must be indexed on the field C_BEN_CODE. Therefore, the records are in alphabetical order by the Benefit Code. When you initialize the class table, you look at the first record in the table. If the Benefit Code is ‘ADD’, ‘LTD’, or ‘WI’, you define the appropriate variable as true and then skip to the next record using FORCE NEXT. You can skip to the next record because the next record must have a Benefit Code is alphabetically after the current record.

Example:
```
DEFINE benadd .F.
DEFINE benltd .F.
DEFINE benwi .F.
INITIALIZE class, "db_classes"
IF c_ben_code = "ADD"
DEFINE benadd .T.
FORCE NEXT
ENDIF
IF c_ben_code = "LTD"
DEFINE benltd .T.
FORCE NEXT
ENDIF
IF c_ben_code = "WI"
DEFINE benwi .T.
FORCE NEXT
ENDIF
NEXT class
```

GENERATE

Syntax: GENERATE Modeldocument_UISR, IStreamdocument_UISR

This rule generates a particular IStream document to create a mass mailing or multiple sub-generations of a model document. If you compare it to the mail-merge operation in Word, the word processor creates a single document, with many individual pages as separate letters in one large file. The GENERATE rule can create individualized documents from several component pieces each in their own separate files.

In the above syntax:

- `IStreamdocument_UISR` is the Universal Information Source Reference (`InfoSourcename:itemreference`) for the generated document,
- `Modeldocument_UISR` is the UISR which the model document is to generate from: this syntax tells Assembler which model document to generate from, and where to put the assembled IStream document.

You can use the GENERATE rule with a default InfoSource so that you do not need to specify the `InfoSourcename` in the `Modeldocument_UISR` and the `IStreamdocument_UISR`.

Example: You generate a document to be stored in `C:\abc\def`, where `C:\abc` is a configured InfoSource named *Genlocation*, and the model document it is generated from contains the following rule:

```
GENERATE MyModels:"Model2", Genlocation:"def\doc2.clg"
```

In this example document *doc2.clg*, generated as a result of executing GENERATE, is stored in *C:\abc\def\doc2.clg*. Generation by this rule passes back only the values of the persistent variables.

IF

Syntax: IF conditional expression

The IF rule is used to execute code based on whether a conditional expression evaluates to TRUE or FALSE. Its general syntax is:

IF conditional expression
ELSEIF conditional expression *(optional)*
ELSE *(optional)*
ENDIF

where the conditional expression can be a logical (Boolean) variable, an expression or a function.

IF with Multiple Conditions

You can specify more than one condition in an IF statement. To do this, use either .AND. or .OR. to join the conditions.

In this example, different wordings apply depending on the value of the p_state variable. Assembler evaluates IF, ELSEIF and ELSE to determine which wording should be included in a generated IStream document.

Example:
```
IF p_state == "TX" .OR. p_state == "FL"
TEXT
The maximum can be no greater than $1,000,000.
ENDTEXT
ELSEIF p_state == "AL" .OR. p_state == "CA"
TEXT
The maximum can be no greater than $500,000.
ENDTEXT
ELSE
TEXT
The maximum is $50,000.
ENDTEXT
ENDIF
```

If the wording applies to a specific state under specific conditions, you can also use an IF statement.

Example:
```
IF p_state == "TX" .AND. p_pregnancy = .T.
TEXT
Special Pregnancy wording for Texas
ENDTEXT
ELSE
TEXT
Wording for everything aside from pregnant Texans.
ENDTEXT
ENDIF
```

You can also combine the use of .AND. and .OR. in one IF statement. When you do, you must group the conditions so that the meaning is correct.

If you want printing of the wording to occur only if the `p_pregnancy` variable is True and the state is either Texas or Florida, group the conditions as follows:

Example:
```
IF (p_state == "TX" .OR. p_state == "FL") .AND.
p_pregnancy = .T.
TEXT
The maximum can be no greater than $1,000,000.
ENDTEXT
ELSE
TEXT
The maximum is $50,000.
ENDTEXT
ENDIF
```

If you want the wording to print if the state is Texas regardless of whether pregnancy is selected, or if the state is Florida and pregnancy is True, group the conditions.

Example:
```
IF p_state == "TX" .OR. (p_state == "FL" .AND.
p_pregnancy = .T.)
```

```
TEXT
The maximum can be no greater than $1,000,000.
ENDTEXT
ENDIF
```

IF with Nesting

If necessary, you can enclose, or nest, an IF statement within another IF statement.

Because nested IF statements can become quite complicated, you should include on-screen comments. For more information, see *COMMENTS* on page 54.

You should also indent each nested statement to make it easier to find when checking your document. However, do not indent the text you want printed or it will appear indented on the printed page.

Important: Ensure that the second statement is entirely nested within the first, and that there is no conflict between statements.

Example:
```
IF c_life = .T.
  IF !EMPTY(c_flatamt)
  TEXT
Lifetime maximum <dol_amt(c_flatamt)>
  ENDTEXT
  ENDIF
ENDIF
```

If this class has life coverage, and if there is a flat amount of insurance, this wording is printed:

Lifetime maximum $50,000.00

Using the ELSEIF rule

The ELSEIF rule is an optional rule for use within IF...ENDIF. ELSEIF is useful when you have more than two conditional wordings that could apply.

Example:
```
IF c_State == "TX"
   ***TX information
ELSEIF c_State == "NY"
   ***NY information
ELSEIF c_State == "CA"
   ***CA information
ELSE
   ***all other states
ENDIF
```

Using the ENDIF rule

Complete the IF rule with ENDIF or it will produce errors.

Example:
```
IF gender = "F"
TEXT
Female
ENDTEXT
```

```
ENDIF
```

IStream Document Manager uses this wording if the value of the variable `gender` is F.

Example:
```
IF !Empty(plan)
TEXT
Since you selected our Gold Service plan, you are also
entitled to the additional services and benefits outlined in
Appendix C.
ENDTEXT
ENDIF
```

If you want to insert variable wording within text, use a Variable IF Statement. For an example of IF used as a function, see *IF* on page 114. To establish different instructions for different situations, use an IF statement.

INCLUDE

This rule supports the following syntaxes.

Syntax: INCLUDE InfoSourcename:"itemreference"

The INCLUDE rule references a section so that it can be included (that is, code will be executed and content will appear) in the assembled document. The syntax takes on various forms depending on what you want to include.

Included files can be designated to consist of text only, or rules and text. If the section you want to include has only text, select the **Text Only Section** box in **Section Properties** for that section.

In general, `InfoSourcename` and `itemreference` can be string expressions. `InfoSourcename` may specify a file system InfoSource name pointing to a directory in the file system, or an IStream Document Manager Repository InfoSource name pointing to a folders in the DMS that contains the referenced section (model document section) specified in `InfoSourcename`. You can omit `InfoSourcename` in the INCLUDE rule if the corresponding `InfoSourcename` was specified in a preceding USEIS rule.

Use the INCLUDE rule as many times as necessary to include required sections.

You can use variables in an INCLUDE rule but ensure they are in brackets.

Example:
```
INCLUDE IStreamFS:("Dependents\" + PolicyID + ".CDS")
```

where **`PolicyID`** is a string variable defined before the INCLUDE rule.

The **Author's Reference Wizard** can help you build references to use with the INCLUDE rule. From the **Author** menu, choose **Tools**, **Authoring Assistance**, then **Reference Wizard** from the menu or click the **Reference Wizard** toolbar button.

Example: **Including a section from a file system:** In this example, all model document sections are files stored on a file server. Each file has a unique name. During document assembly, you want to include a section called `testsect.cds` from a

configured file system type InfoSource named `Testing` that points to the directory where `testsect.cds` is stored.

The code would be:

```
INCLUDE Testing:"testsect.cds"
```

Example: **Including a section from a document management system when a variable evaluates to "true".** Your sections or model documents are stored in a DMS repository named `dms_tng`. During document generation, you want to include a section called `ADD` if the variable `ADDCVG` evaluates to "true". The section may be anywhere in the repository. The `itemreference` must include the path of folder names and the section's document name.

The code would be:

```
IF ADDCVG = .T.
INCLUDE
dms_tng:"\Livelink\Logins\dkenning\Documents\ModDocs1\InsCoy
\ADD"
ENDIF
```

Syntax: INCLUDE folder|section_file, InfoSourcename

This is similar to the first syntax, however, the section is not referenced by name but by the corresponding file name.

The folder is the name of the directory (for a file system InfoSource) that contains the section, and the file name can be a string expression. The `InfoSourcename` represents the name of the InfoSource that contains the section. The `InfoSourcename` is optional if it is already defined with the USEIS rule.

When generating, the first section that matches the criteria is located and the appropriate information is included in the generation.

Example: **Pointing to a directory in a file system InfoSource:** A file system InfoSource named `Scripts` refers to a directory: `S:\Enterprise Workspace\Documents`. Included in this directory is the subdirectory `Testmod1`, which has the files of the model document's sections. The coding would be:

```
USEIS Scripts
INCLUDE "Testmod1"|"testsect.cds"
```

Syntax: INCLUDE folder||section_name, InfoSourcename

This form of the INCLUDE rule is used to select from multiple sections based on an `effectdate` when working with effective and termination dates. This minimizes the requirements for extra coding (the requirement for IF statements).

This syntax assumes that the section is stored as a document in the DMS and has the following attributes:

- Section Name (different from the document name)
- Effective Date
- Termination Date

In addition to these attributes, the following requirements must be met:

- the IStream Document Manager type must be used for the InfoSource (the || (double pipe) is not supported by the File System InfoSource)
- multiple sections having the same section name must not have overlapping effective and termination periods

Example: **Using || with Effective Dates:** In the DMS, there are two sections (with the same section name) stored in the model document's folder, `My Model`. These sections have the following properties.

Property	First Section	Second Section
Document Name	My First	My Second
Section Name	My Section	My Section
Effective Date	01/01/2007	07/01/2007
Termination Date	06/30/2007	12/31/2007

The path to the folder containing the model document `My Model` is specified by the InfoSource Models. Therefore, the INCLUDE statement can be coded as follows:

```
INCLUDE "My Model"||"My Section","Models"
```

Syntax: INCLUDE "**.\directoryname\section_file**"

or

INCLUDE "**..\..\directoryname\section_file**"

This INCLUDE rule is used when sections of your model document are not all located in the same folder or directory.

- ".\" indicates the current directory or folder
- "..\" indicates the parent directory or folder

Note: This syntax should be used when you create model documents intended for use with the **Edit** function in IStream Document Manager.

Example: You have a master document with subsections in the InfoSource `LocalFS`:

- the master document is saved as `dir1\dir2\dir3\Master.cms`
- section 1 is saved as `dir1\dir2\dir3\Section01.cds`
- section 2 is saved as `dir1\dir2\dir3\dir4\Section02.cds`
- section 3 is saved as `dir1\dir2\Section03.cds`
- section 4 is saved as `dir1\Section04.cds`

The INCLUDE rules are coded as follows:

INCLUDE Rule	Description
INCLUDE ".\Section01.cds"	`Section01.cds` is in the same directory as `Master.cms` ".\" makes the relative path begin in the directory where Master.cms is located
INCLUDE ".\dir4\Section02.cds"	`Section02.cds` is in a directory beneath the directory containing `Master.cms` ".\" makes the relative path begin at the directory containing `Master.cms` `dir4` specifies the name of the directory containing Section02.cds
INCLUDE "..\Section03.cds"	`Section03.cds` is in the parent directory of the directory containing `Master.cms` "..\" makes the relative path begin at the parent directory of the directory containing `Master.cms`
INCLUDE "..\..\Section04.cds"	`Section04.cds` is two directories higher than the directory containing `Master.cms` "..\..\" makes the relative path begin two directories above the directory containing `Master.cms`

FileSystem and IStream Document Manager InfoSources Differences

When writing INCLUDE rules for model documents that will be used with IStream Document Manager InfoSources, note that the directory structure for an IStream Document Manager InfoSource may be different from that of a FileSystem InfoSource.

In an IStream Document Manager InfoSource, the master section may be located in a model document folder that has the same name as the master section.

For example, the master section named **MyModel** may be located in IStream Document Manager InfoSource as follows:

`"IStreamDM":"dir1\dir2\dir3\MyModel\MyModel"`

The first reference to `MyModel` is to the folder containing the master section. The second reference to `MyModel` is to the master document itself.

Example: You have a master document with subsections in the InfoSource `LocalFS`:

- the master document is saved as `dir1\dir2\dir3\Master.cms`
- section 1 is saved as `dir1\dir2\Section01.cds`
- master section X is saved as: `dir1\SectionX\SectionX.cms`

In these examples, the InfoSource name is omitted. The INCLUDE rules are coded as:

INCLUDE Rule	Description
INCLUDE "..\Section01.cds"	Section01.cds is in the parent directory of the directory containing Master.cms. "..\" makes the relative path begin in the parent directory of the directory where Master.cms is located.
INCLUDE "..\..\SectionX\SectionX.cms"	SectionX.cms is two directories above the directory containing Master.cms and is associated with the model document “SectionX”. "..\..\" makes the relative path begin two directories “above” the directory containing Master.cms. SectionX specifies the name of the model document/directory containing SectionX.cms. SectionX.cms specifies the section to be included.

Adding a Microsoft Word Document to the Model

You can use the INCLUDE rule to add a Microsoft Word document to the model. When you generate the model, the contents of the new Microsoft Word section will be included in the resulting IStream document. The INCLUDE syntax for adding a Microsoft Word document to the model is the same as the INCLUDE syntax used to add an IStream section.

Method: INCLUDE a Microsoft Word document in your model

1. Use an INCLUDE rule to refer to the section. See *INCLUDE* on page 67 for the syntax.
2. From the tree view, select the model document section that you added the Microsoft Word document to. Click  .

   The Microsoft Word section is now visible in the tree view as uncompiled, with a warning icon next to it. When you generate the model, the contents of the Microsoft Word section will be included in the IStream document as text.

   You can open your section by double-clicking it in the tree view. If you save the section, it will be automatically converted to an IStream section. You cannot use the **File > Open Section...** command until you have converted the section to an IStream section. If you want Assembler to process rules and functions in the section, you must also convert it to an IStream section.

Method: Convert the Microsoft Word section to an IStream section

1. Click **Author > Rebuild Model Document**

   or

   Double-click the section in the tree view to open it. Click **File > Save**.

   The section will now be treated the same as any other IStream section. You can use the **Open Section...** command to view and make changes to your section.

Note: Although the Microsoft Word section has been converted to an IStream section, it will retain its .DOC file extension.

2. By default, the new section is included as a **Text Only** section. If you want IStream Assembler to process rules and functions in the section during generation:

   a. Open the section and click **File > Section Properties**.

   b. Clear the **Text Only Section** check box, and then click **OK**.

Important: If the Microsoft Word document added to the model was originally saved to the DMS, it remains a Microsoft Word document in the DMS. You will not be able to use the **Author** command in the DMS to open your section. If you want to open your section in Author, use the **File > Open Section...** command from within Author instead.

Using the FORCEINCLUDE Rule

The FORCEINCLUDE rule is a special form of the INCLUDE rule. When you use FORCEINCLUDE to incorporate an IStream section in your model document, effective and termination dates applied to the section are ignored and the section is always included in generated IStream documents.

Because FORCEINCLUDE ignores the effective and termination dates applied to your IStream sections, overusing it may compromise the control that you have over document wordings and when they are included. To make sure that the content of your IStream documents is correct and that your model documents are sustainable, you should generally use the INCLUDE rule in combination with logical rules (IF, ELSEIF, ELSE, ENDIF), or modify the effective and termination dates to set conditions for the inclusion of sections.

References to sections use the same syntax in the FORCEINCLUDE rule as they do in the INCLUDE rule. For example, to FORCEINCLUDE a section called `coverage_details.cds` from a file system InfoSource called `SectionFolder`, you would use the following code in your model document:

```
FORCEINCLUDE SectionFolder:"coverage_details.cds"
```

INITIALIZE

Syntax: INITIALIZE tablename, "InfoSourcename"

Use the INITIALIZE rule to find all matching records in the table for document assembly. This is very useful for multiple class or multiple coverage records.

This rule is used only with a UserDB type of InfoSource. It is used to loop through tables in the database.

The INITIALIZE rule instructs Assembler to read the first record from the named database table that matches the key in the work record.

The NEXT rule instructs Assembler to read the next record in the resulting rowset, and to continue until it reads all of them.

Syntax: INITIALIZE tablename, "InfoSourcename"

`Tablename` is the name of a table from the mapping in the UserDB InfoSource. `InfoSourcename` identifies a UserDB type of InfoSource. `InfoSourcename` can be omitted if the INITIALIZE rule is preceded with the USEIS rule and the correct UserDB InfoSourcename.

Note: Remember to complete the rule with NEXT when you use INITIALIZE.

Example: You may want to print a list in your final document showing the maximum coverage available for each class of employee with policy number 12345678. To do so, use the initialize rule with the following sample `class.dbf` table:

c_pol_no	c_class_no	c_class_ds	cmax_covg	class_age	c_life
11223344	A	All	100000	65	T
11223344	B	Retired	50000	70	F
12345678	A	Executives	50000	65	T
12345678	B	Office Staff	30000	65	T
12345678	C	All others	15000	65	F

Use the TEXT...ENDTEXT rule to set up the headings for the list you want to print. Use the QUERY rule to select the database table.

Set the TEXT...ENDTEXT rule to print the classes.

Use variable insertions to print the information that you want in your list (in this example, the c_class_no field, the c_class_ds field and the amount of the cmaxcovg field).

Use the NEXT rule to instruct the system to find the next record which fulfills these requirements, and to continue until it finds all of them:

```
TEXT
Class  Description Maximum Coverage
ENDTEXT
USEIS user_db
QUERY "select * from class"
```

```
IF c_pol_no=12345678
TEXT
<c_class_no> <c_class_ds> <DOL_AMT(cmax_covg)>
ENDTEXT
ENDIF
NEXT
UNUSEIS
```

This wording is printed in the assembled document:

Class	Description	Maximum Coverage
A	Executives	$ 50,000
B	Office Staff	$ 30,000
C	All others	$ 15,000

Notice how an IF statement can be used inside the INITIALIZE...NEXT loop to select only those records that match specific criteria. In the above example, only records with a policy number of 12345678 were chosen.

INITIALIZE with TEXT and IF statement

Sometimes you may want to print information from multiple records on a single line. You may want the values of a variable within a sentence, separating each value with a comma, such as when you list multiple class descriptions. There are two methods to do this.

Example: **Method 1:**

```
DEFINE Records 0
INITIALIZE class, "userdb_GroupIns"
DEFINE records (Records + 1)
NEXT class
DEFINE Count 0
INITIALIZE class
DEFINE Count (count + 1)
IF ((count != records) .and. (count != (records -1))
TEXT
<trim(c_class_ds)>,
ENDTEXT+
ENDIF
IF (count = (records -1))
TEXT
<trim(c_class_ds)> and
ENDTEXT+
ENDIF
IF (count = records)
TEXT
<trim(c_class_ds)> qualify.
ENDTEXT+
ENDIF
NEXT
```

The assembled wording looks like this:

Executives, Office Staff, and All Others qualify.

Example: **Method 2:**

This is the second method for listing values of a variable within a sentence, separating each value with a comma, such as when you list multiple class descriptions:

```
DEFINE classes ""
*** sets up a blank defined variable
IF LEN(classes) = 0
DEFINE classes (TRIM(c_classnum))
*** adds the first class to the defined variable
ELSE
DEFINE classes classes + ", " + TRIM(c_classnum)
*** adds other classes to the defined variable,
*** and separates them with commas
ENDIF
IF LEN(classes) > 2
*** checks to see if more than 1 class
*** is covered.
DEFINE classlist "Classes " + LEFT(classes,RAT(", ",classes)
-1) + " and " + SUBSTR(classes,RAT(", ",classes) + 2)
ENDIF
TEXT
This coverage applies to <classlist>.
ENDTEXT
```

NEXT

Syntax: NEXT tablename

Use the NEXT rule to close the INITIALIZE and QUERY rules. For examples, see *INITIALIZE on page 73* and *QUERY on page 75*.

QUERY

Syntax: QUERY "SQL SELECT Statement", "Database InfoSource name"

Author Queries are used to communicate with databases using embedded SQL Select statements within the QUERY/NEXT rule.

The Query statement extracts the data from the database to make it available for the model document.

Query statements are also used to create table relationships that establish the data flow for the model document.

Connect to an ODBC Database InfoSource Using Queries

ODBC Database InfoSources do not contain mappings. This means that the table relationships are not established in the InfoSource (as they were in the UserDatabase InfoSource). A model document still needs to contain the database table relationships to make data available for generation.

The QUERY/ NEXT rule has two purposes:

1. To make data available to the model document.
2. To create relationships between the tables that are needed for the model document.

Elements of a Query Statement

There are three essential parts of every query statement:

1. The IStream Author Query rule.
2. The SQL select statement that is embedded inside the Query rule.
3. The database InfoSource name.

Example:
```
QUERY "select* from tablename where fieldname = 'A'",
"isname"
    NEXT
```

Note: The data from the table is only available between QUERY/NEXT. Also, the NEXT statement closes off the data for the query and creates a loop.

SQL Select Statement with String Data

The Query statement parses out a SQL compliant select statement that is sent to the database and looks like this:

```
select * from tablename where fieldname ='A'
```

Note: SQL reads string data with single quotes.

The asterisk (*) selects all fields from the table and brings the result set into memory for the model. The fieldname element is the value that needs to match so the data is brought in for the model.

Optional Element of a Query Statement – BREAK

Using QUERY / NEXT will result in looping through the selected database table. When looping is not required, BREAK should be used before NEXT.

Example:
```
QUERY "select * from tablename where fieldname
= 'A' ", "isname"
    BREAK


    NEXT
```

Note: The data from the table is only available between QUERY/NEXT. The BREAK statement is optional but the NEXT statement is required.

Role of Double Quotes within a Query Statement

Double quotes are used to stop and start the IStream Author language as opposed to the SQL select statement.

The single quotes are only used when the comparing value is a string value.

Example:
```
QUERY "select * from Branch where BRNUMB= " +
"'" +BRNUMB+ "'", "isname"


    NEXT
```

Note: The comma separates the parameters in the query.

This example selects all fields from the `Branch` table. Where the `BRNUMB` value matches the `BRNUMB` requested, the data/result set is brought into memory.

Role of the + Sign within a Query Statement

This SQL statement is comparing string data, which needs single quotes within the SQL select statement (other examples follow).

For a query statement using numeric data, see *Query Statement using Numeric Data (Str)* on page 78.

Example:
```
QUERY "select * from CLAIMS where CLAIMCODE ="+"'"
+CLAIMCODE+ "'", "isname"


    NEXT
```

Note: The SQL statement is embedded within the query. The plus signs are used to concatenate SQL language with IStream Author language. The SQL select statement reads string data with single quotes surrounding the value.

The preceding example selects all fields from the `Claims` table. Where the `ClaimCode` value matches the `ClaimCode` requested, the data/result set is brought into memory.

Query Statement using String Data

In the following example, the `POLICYNUMB` field in the `POLICYHOLDER` table is a string data type:

Example:
```
QUERY "SELECT * FROM CLAIMS WHERE POLNUMB=" + "'" +
POLICYNUMBER + "'", "ISNAME"


    NEXT
```

The following examples shows an alternate way to code a query statement with string data:

Example:
```
QUERY "SELECT * FROM CLAIMS WHERE POLNUMB='" + POLICYNUMBER
+ "'", "ISNAME"
```

```
    NEXT
```

A query can be built in different ways when dealing with string data.

Up to and including the = sign, the Query statement is the same in both scenarios. The second example reduces the amount of coding and concatenation that occurs in the Query statement.

Note: Both methods produce the same SQL statement that is extracted for the model document as follows:

```
SELECT * FROM CLAIMS WHERE POLNUMB='10'
```

The POLNUMB is 10 but it is a string value, not a numeric value, within the database.

Query Statement using Numeric Data (Str)

The Policyholder BRNUMB field and the Branch BRNUMB field are both numeric data type fields within the database.

Example:
```
Query "SELECT * FROM BRANCH WHERE BRNUMB=" +STR(BRNUMB),
"ISNAME"
    NEXT
```

The role of the plus (+) sign is still to concatenate both the SQL and the IStream Author languages into one statement.

The STR function converts numeric data to a string data type. The entire query line must be string for IStream Author to evaluate it.

The STR function is needed within the query so that the system can interpret the field as being a string, and concatenate the numeric data. If the STR function is not used, numeric data is added, not concatenated.

The STR function is an IStream Author function.

SQL Select Statement With Numeric Data

If the data in the table is a numeric field, the SQL statement looks like this:

Example:
```
select * from BRANCH where BRNUMB = 10
```

Using QUERY with IStreamXML InfoSources

When you are querying an IStreamXML InfoSource, you need to use two separate query statements. The first query initializes the InfoSource, while the second query retrieves the information.

When using QUERY with an IStreamXML InfoSource, you must place a BREAK statement before the QUERY's closing NEXT rule for the first query.

Example: In the following example, Automobile Data is the InfoSource name.

```
QUERY "", "Automobile Data"

QUERY "SELECT * FROM Automobile WHERE PolicyID = 1111",
"Automobile Data"

NEXT
```

```
BREAK
NEXT
```

Using QUERY with UserDatabase InfoSources

When using QUERY/NEXT SQL statements in a model document that uses both a UserDatabase type of InfoSource and INITIALIZE/NEXT statements, note the following condition:

- If the QUERY/NEXT statement references the same database tables that exist within the UserDatabase mapping, then the QUERY/NEXT statements should not be placed between the INITIALIZE ROOT statement and its corresponding NEXT statement.

The QUERY/NEXT statement should be placed outside the INITIALIZE ROOT/NEXT coding. DEFINE statements or ARRAYS can then be used to store the data for reference within the model document.

Performing Different Queries Against Same Database Table

In the following example, there is a database InfoSource `GroupPol` containing the table `CUSTOMERS` and the column `COMPANY` in that table. You are making two different queries (one nested) against this table, each producing result sets containing the column `COMPANY`. Each result set will contain different values.

The scope of the QUERY rule for defining a default InfoSourcename is effective as long as code is enclosed within the QUERY/NEXT rule. This means that each UISR that references a rowset produced by this query can be used without an InfoSourcename within the scope of this rule, except in ambiguous cases.

Example:
```
QUERY "Select * from CUSTOMERS where XYZ=A", "GroupPol", "A"
QUERY "Select * from CUSTOMERS where XYZ=B",,"B"
TEXT
This references <.A:COMPANY> from rowset A,
while this references <.B:COMPANY> from rowset B
ENDTEXT
NEXT
NEXT
```

Note: A period must precede the rowset name.

SELECT

Syntax: SELECT tablename, userdb_InfoSourcename

Use SELECT when you need information from only one table. The SELECT rule enables you to look at one specific table without incurring automatic initialization of other tables. This improves the speed of your document generation if the table has child tables or sub-elements. A user database mapping is hierarchical. That is, it is organized such that one table is said to be above, below, or on the same level as other tables. Each level indicates a level of dependency.

Note: The SELECT rule applies only to the User database type of InfoSource and is only used in conjunction with the INITIALIZE...NEXT.

Whenever IStream Document Manager initializes a table in a database (for example, during and INITIALIZE...NEXT loop), it initializes all dependent tables in the database (all the tables below the selected table). This process can take some time. To optimize generation time, you can SELECT a specific table, if you know the field or fields that you require are in only one table.

Use the following syntax:

```
SELECT tablename, userdb_InfoSourcename
```

where `tablename` is the name of a table in the user database. It is a mandatory parameter and can be a valid name, that is, one which is not evaluated; `userdb_InfoSourcename` is the name of the user database InfoSource (string constant). It can be omitted if a default is previously defined. If the InfoSourcename is indicated in the SELECT rule, it is not treated as a default InfoSource.

Example:
```
TEXT
This policy provides benefits for the employees of the
following organizations:
ENDTEXT
SELECT division, "userdb_Life"
QUERY division
TEXT
<div_name>
ENDTEXT
NEXT
SELECT ALL, "userdb_Life"
```

SELECT ALL

Syntax: SELECT ALL, userdb_InfoSourcename

`Userdb_InfoSourcename` is an optional parameter if a default has been previously defined.

SELECT ALL is a non-mandatory matching rule for the SELECT rule and directs IStream Document Manager to return to processing all tables in a user database InfoSource. Both SELECT and SELECT ALL rules can only be applied to a user database InfoSource type.

Example:
```
TEXT
This policy provides coverage for the following perils:
ENDTEXT
SELECT ptype, "userdb_Coverage"
QUERY ptype, "userdb_Coverage"
TEXT
<per_name>
ENDTEXT
NEXT
SELECT ALL, "userdb_Coverage"
```

TEXT

Syntax: TEXT

The TEXT...ENDTEXT rule is one of the most important rules to know. It instructs IStream Document Manager to treat all text and formatting codes between the TEXT...ENDTEXT as literal text, and to insert it into the document so that it is printed out exactly as it appears on the page.

Because TEXT...ENDTEXT is the most common rule used, it is also the most frequent source of errors. Remember that all text required in the assembled document must appear between these codes. All the formatting or other word processing information must also be placed between TEXT...ENDTEXT.

Note: You may not nest TEXT...ENDTEXT blocks.

To place an expression, including variables or a UISR within TEXT sections, you must use angle brackets < >, such as <variable>. This tells Assembler to read that word as a value instead of plain text.

The TEXT rule indicates to Assembler that word processing text and codes have begun. The ENDTEXT rule tells IStream Document Manager that text and codes have ended, and that it must interpret rules again, so always be sure that every text block has a corresponding ENDTEXT.

If you do not have a closing ENDTEXT statement to correspond to the TEXT rule, all the text from the model document (including the next rules and functions) are printed in the assembled document as text. IStream Document Manager continues processing as text until it encounters an ENDTEXT rule.

Every TEXT and ENDTEXT rule must be followed by a hard return (¶). However, you can let the body of text that you type wrap automatically to the next line (a soft return).

Text Formatting Notes

A generated IStream document displays whatever is between TEXT and ENDTEXT exactly as it appears in your model document. For example, if you indent text, apply bullets, change fonts, or include symbols, your output text includes these attributes.

Formatting applied in a text block that will be concatenated to the next text block may affect the following the text block. For example, if you change text in a text block to be bold and italic, the text following the text block may inherit the bold and italic setting, even though it is not formatted this way in the model document section.

Example: In this example, the ¶ paragraph marks are displayed for reference. In this code, the first part of the sentence should be bold and italic, but the second part should be plain text.

Note that following examples apply to both the `ENDTEXT` and `ENDTEXT+` rules.

`TEXT`¶

This text should be bold and italic; ¶

```
ENDTEXT +¶
TEXT¶
```

this text should be plain.¶

```
ENDTEXT¶
```

The output for this example will be:

This text should be bold and italic; this text should be plain.

To fix this, select only the ¶ paragraph marker at the end of the content of the first text block and apply the required formatting. In this example, select the ¶ paragraph marker at the end of:

This text should be bold and italic; ¶

and apply the required formatting to the ¶ paragraph marker. Text following the marker will inherit the formatting assigned to the ¶ paragraph marker:

This text should be bold and italic; this text should be plain.

Example: In this example, the first part of the sentence should be bold and italic, but the second part should be plain text.

```
TEXT
```

Policy Holder Name	***Policy Holder Address***

```
ENDTEXT+
***The text in the above table is bold and Italicized
```

Query "Select* from Policy", "PolData"

```
TEXT
```

<PolName>	<PolAdd>

```
ENDTEXT+
NEXT
```

The text in the above table will be concatenated to the TITLE row of the table and should be plain text without bolding the italics.

Important: If you change the formatting of a text block (such as the font name, size, or bolding), ensure that you select only the text *inside* the text block when formatting it. In other words, ensure that the `TEXT` rules use different font properties than the content of the text block to avoid the format spilling into the next text block.

Table of Contents Fields in TEXT...ENDTEXT Blocks

When including a Word table of contents field in a TEXT...ENDTEXT rule, either additional text or a hard return must be between the table of contents and the ENDTEXT statement.

Example: In this example, paragraph marks and Word field codes are displayed. Note the hard return (the ¶ paragraph mark) inserted between the table of contents field and the ENDTEXT statement:

```
TEXT¶
{ TOC \o "1-3" }¶
¶
ENDTEXT¶
```

TEXT...ENDTEXT Examples

Example: This portion of a contract is to be printed exactly as shown.

```
TEXT
Premiums
Premiums are payable by the Policyholder in advance at the
Insurance Company's Head Office or to its authorized agent.
ENDTEXT
```

Example: This text block has variables enclosed in the text block. Remember that variables must be enclosed in angle brackets to distinguish them from plain text.

```
TEXT
When contacting the company about your claim, be sure to
include your policy number in your correspondence. Your
policy number is <p_pol_no>. A copy of your correspondence
should also be sent to <bkr_name>, <bkr_addr>. This will
ensure that you receive a timely response.
ENDTEXT
```

Note: Do not use variables within text boxes contained inside tagged text blocks, as these variables cannot be customized in Customizer.

Example: Although you cannot nest TEXT...ENDTEXT blocks, TEXT...ENDTEXT pairs within other nested rules are supported. Remember, do not indent the nested text, or it prints indented.

```
TEXT
Eligibility Wording
ENDTEXT
IF state == "CA"
*** California exception
TEXT
Residents of California are not eligible for this benefit
ENDTEXT
ELSE
*** All other states
TEXT
Residents in the state of <state> are eligibile for this
benefit.
ENDTEXT
ENDIF
```

TEXT with Variables

When necessary, you can insert, or embed, a field in your text. An embedded field is data in your database that changes, such as a policy number, and which is inserted within a block of text.

A variable is surrounded by delimiters, usually angle brackets (< >). IStream Document Manager treats all text between the delimiters as a variable, and evaluates it. The result of this evaluation appears in the assembled document.

Example:
```
TEXT
The policy number is <p_pol_no>.
ENDTEXT
```

The wording in the assembled document is:

The policy number is 1234567.

TEXT with Accented Characters

If you need to use special characters in a variable if statement, such as é, â or ç, use the **Insert Symbol** menu command or the keyboard shortcut to insert the accented character into the expression. To see the available characters, select **Start > Programs** > **Accessories** > **System Tools** > **Character Map**. For example:

```
<If(drug_ben = .T., "vous avez une allocation pour
médicaments"," ")>
```

The French phrase in the assembled document is:

vous avez une allocation pour médicaments

OR:

If you need to use special characters in an expression, such as é, â or ç, you can use `CHR(`*`ASCII or Unicode value`*`)` to refer to the character. The following example uses the ASCII value to return the é character:

```
<If(drug_ben = .T., "vous avez une allocation pour m" + chr(
233 ) + "dicaments"," ")>
```

The French phrase in the assembled document is:

vous avez une allocation pour médicaments

TEXT with Language Settings

(TEXTE, TEXTF, TEXTG, TEXTS, and so on)

When you use the text rule, IStream Document Manager processes all text blocks, regardless of their language. However, you may want your document to be printed in one of several languages, depending on a database variable. In this case, you want to insert text in a specific language.

Note: To specify a language, you use text blocks with a language setting.

As a default, IStream Document Manager brings in all TEXT...ENDTEXT sections. To change the default insert the SETLANGUAGE function, see *SETLANGUAGE* on page 139.

Example:
```
TEXTE
This only appears in English documents.
ENDTEXT
TEXTF
```

```
This only appears in French documents.
ENDTEXT
TEXTG
This only appears in German documents.
ENDTEXT
TEXTS
This only appears in Spanish documents.
ENDTEXT
TEXT
This appears in all documents, regardless of language.
ENDTEXT
```

TEXT With Array References

Because array references must use square brackets (for example, `<myArray[1,1]>`), you cannot use them in `TEXT []…ENDTEXT` blocks or in **Text Only** sections that have the bracket style set to [ ].

TEXT {}

Syntax: TEXT {}

Angle brackets < > are the default brackets used to denote variables within a text block. However, you may sometimes want to use either < or > to mean less than or greater than. In this case, you do not want IStream Document Manager to interpret the angle bracket as an operation.

To avoid this, you can change the default insertion brackets to brace brackets { } or square brackets [ ]. IStream Document Manager interprets the braces as the delimiters for variables for this text rule only. For the next text rule, the delimiters revert to being angle brackets unless you again use TEXT {}.

Example:
```
DEFINE AGE 55
TEXT {}
{IF (AGE < 65, "Employees are covered to the earlier of age
" + STR(AGE) + " or retirement.", "Employees are covered to
age 65.")}
ENDTEXT
```

Note: Text variables revert to using the default angle brackets after ENDTEXT.

Generated result: Employees are covered to the earlier of age 55 or retirement.

TEXT DEFAULT {}

Syntax: TEXT DEFAULT {}

The TEXT DEFAULT {} rule is similar to the TEXT{}. It also enables you to change the default delimiters, which are angle brackets < >, to another type of bracket [ ] or { }.

However, unlike the TEXT{} rule, the TEXT DEFAULT {} stays in effect for the entire section. Other sections included in the model document or section still use

angle brackets as delimiters, unless you use the TEXT DEFAULT {} rule there also.

Example:
```
TEXT DEFAULT {}
The policy number is {p_pol_no}.
ENDTEXT
TEXT
The state is {p_state}.
ENDTEXT
```

TEXT DEFAULT {} Rule within IF Statements

If you nest the TEXT DEFAULT {} rule within an IF statement, the delimiters defined in the TEXT DEFAULT {} rule will apply throughout the section, even when the conditions for the IF statement are not met.

Example:
```
DEFINE var 0
IF var = 1
TEXT DEFAULT {}
ENDTEXT
ENDIF
```

You would expect the default delimiters < > to apply to the TEXT rules in the section, because the condition for the IF rule is not met. var = 0, not 1 as required for the IF rule to be executed. However, in the TEXT rule that follows:

```
TEXT
Your coverage amount is <coverage_amt>
{For more information about your coverage, please contact
Towne Insurance}.

ENDTEXT
```

<coverage amount> is not evaluated as a variable like expected. Instead, IStream Document Manager tries to evaluate the sentence within {} as an operation. IStream Document Manager interprets {} as the delimiters according to the TEXT DEFAULT {} rule, even though the IF statement conditions are not satisfied.

TEXT TAG

Syntax: TEXT TAG TagName

Assigning a tag name to a text block allows the text inside the text block to be customized in an IStream document from within IStream Customizer. A customizable text block is referred to as a tagged text. From a tagged text block, custom wordings can be created which allow text in an IStream document to be changed or removed during generation. See the IStream Customizer documentation for complete instructions on using Customizer and custom wordings.

The tag name can be any text string. The tag name cannot contain spaces or special characters.

Example:
```
TEXT TAG TaggedTextBlock
Important Note
```

```
There have been changes made to your policy! Please review
and retain for your records
ENDTEXT
```

Using a Variable as a Tag Name

The following example shows how you can use a VARIABLE as TAG name:

Example:
```
DEFINE variable "tag123"
TEXT TAG <variable>
....
ENDTEXT
```

This can be useful if you want to create a unique series of TAGs. If you have a TAG inside a loop, then each resulting paragraph will have the same TAG in Customizer. If you want each paragraph to have a unique TAG in Customizer, you can add a variable to the TAG name.

Example:
```
DEFINE Count 1
DO WHILE Count < 10
TEXT TAG <count>

The number is <count>...

ENDTEXT
DEFINE Count Count +1
ENDDO
```

This will create nine paragraphs, each with a unique TAG.

UNUSEIS

Syntax: UNUSEIS

UNUSEIS is a mandatory matching rule for USEIS, which cancels the effect of the nearest preceding USEIS rule in the same section. It must be typed in a separate line. Pairs of USEIS/UNUSEIS rules can be nested. See *USEIS on page 87*.

USEIS

Syntax: USEIS InfoSourcename

This rule specifies use of a particular InfoSource ("USE InfoSource") in conjunction with subsequent commands that interact with this InfoSource. By inserting the USEIS rule to establish the connection to a particular database, subsequent INITIALIZE...NEXT or QUERY....NEXT commands implicitly refer to this database. USEIS applies to all types of InfoSources.

In the syntax, InfoSourcename identifies a particular InfoSource.

Even if you specified a default InfoSource, you can use a fully expanded syntax to point to a different InfoSource within your document code. IStream Document

Manager would use that InfoSource location for that section of code and then returns to the default.

Pairs of USEIS...UNUSEIS can be nested.

Example: In this example, USEIS defines `Goldmine` as a user database type InfoSource for all subsequent references to its tables and data. Once the USEIS `userdb_Goldmine` command establishes `userdb_Goldmine` as the InfoSource pointing to the `GOLDMINE` database, there is no further need to use the reference in subsequent commands.

The rule used in a section looks like this:

```
USEIS userdb_Goldmine
```

Functions

A function is a command that performs an action on a variable. For example, you can use a function to format the way data in a field appears in an assembled document, or to perform an arithmetic operation on the value of a variable.

Because functions act on arguments, the following explanations show the format of each function with a sample argument. A function's arguments can not only be variables, but also constants or any valid expression.

Important: When you use the functions in a section, remember to use real arguments instead of the examples shown.

Related Topics

- *Rules and Functions Table* on page 46
- *Authoring Standards* on page 49
- *Rules* on page 52
- *Function Examples* on page 89

Function Examples

For each example of a function, the argument name, its syntax, the format and type of argument, and a sample value are displayed.

How the function is used in a typical fashion in a section is shown, and the result of that function in an assembled document.

Example:
```
argument: amount
type: numeric, 3
value: -123
ABS(amount)
```

Generated result: 123

Important: Angled brackets are used to separate parameters when showing syntax examples. These brackets are not used when entering the function. Therefore, for the following syntax example, `CALL AADD(<arrayname>, <value1>,...,<valueN>)`, you should enter `arrayname` into the function and not `<arrayname>`.

Data Types

The following is a definition of the data types used by IStream Author.

- **Boolean** – a logical value of true (.T.) or false (.F.)
- **Date** – a unique data type that contains a month, day, and year that can be created using functions such as CTOD, DATE, and DMY

Example: `CTOD("01/01/2007") = January 1, 2007`

- **Integer** – Whole numbers, positive and negative.

Example: Numbers from -9223372036854775800 to 9223372036854775800.

- **Float** – all positive and negative real numbers; also known as decimal numbers

Example: Numbers from -922337203685477.87 to 922337203685477.87

- **String** – a set of zero or more consecutive characters within straight quotes

Example: "Smith Corporation"

AADD

Syntax: CALL AADD(<arrayname>, <value1>,...,<valueN>)

Parameter Name	Data Type	Description	Use
arrayname	string	the name of the array to which elements are being added	required
value1	any	a value included in the array	required
valueN	any	a value included in the array	optional

Use the AADD function to add rows to an array. You can add one or more rows to a newly declared array. If your defined array is already initialized (that is, it already has values), then you can only add rows that have the same number of columns as the initialized array, to a maximum of 4,096 rows.

A row can consist of one or more columns. The columns can contain different field types. For example, one may be a numeric value while the next is a character value. An array must be created before the AADD function can be used. An element can be an expression or a function.

Example: You may want to build an array that includes all of the beneficiaries for a plan and their relationship to the insured:

```
DEFINE Beneficiaries {}
QUERY address
IF UPTRIM(addrtype) == "B"
*** this address is for a beneficiary.
CALL AADD(Beneficiaries, {lastname, firstname, relation})
ENDIF
NEXT address
```

For more information on arrays, see *Using Arrays* on page 147.

ABS

Syntax: ABS(<value>)

Parameter Name	Data Type	Description
value	integer or float	a value used to calculate the absolute value

Use this function to calculate the absolute value of an expression. The absolute value of a number is the distance it is from zero (origin).

Example: You need the absolute value of `amount`.

```
argument: amount
type: numeric
1st value: -123
2nd value: 123
ABS(amount)ABS(amount)
```

1st Generated result: 123

2nd Generated result: 123

```
TEXT
The premium amount is <ABS(amount)>.
ENDTEXT
```

The premium amount is 123.

ALLTRIM

Syntax: ALLTRIM(<string_param>)

Parameter Name	Data Type	Description
string_param	string	the value you want to trim leading and trailing spaces from

This function trims all the leading and trailing spaces from a character string.

Example:

```
argument: p_holdnam
type: character
value: "Carpenter  "
ALLTRIM(p_holdnam)
Generated result: Carpenter
TEXT
The policyholder is Mr. <ALLTRIM(p_holdnam)>.
ENDTEXT
```

The policyholder is Mr. Carpenter.

ASORT

Syntax: DEFINE <arrayname> ASORT(<name>,<startlocation>, <numElements>,<row_nameA>,<row_nameB>,<compare_expression>)

Parameter Name	Data Type	Description	Use
arrayname	array	the name of the newly sorted array	required
name	string	the name of the array to be sorted	required
startlocation	integer	the starting element for sort must be numeric if left blank, ASORT uses the first element of the array	optional
numElements	integer	the number of elements to sort if left blank, all elements are sorted	optional
row_nameA	string	the temporary name of the first element passed to comparison must be a character value if left blank, defaults to "X"	optional
row_nameB	string	the temporary name of the second element passed to comparison must be a character value if left blank, defaults to "Y"	optional
compare_expression	string	a comparison expression that controls the order of the sorted results must be a character value cannot be omitted or left blank standard expression is parameter 4 less than parameter 5 example: "X[1]<Y[1]" where X is parameter 4 and Y is parameter 5 For more information, see *Sorting on Multiple Columns* on page 93	required

Use this function to create a sorted array. You first create an array as described in *Creating an Array* on page 148. You then use ASORT to create another array which defines as the sorted array as follows:

Example: `DEFINE BENEFICIARIES {}`

```
... (adding elements to an array)
*** new array that holds the sorted contents of the original
array
DEFINE SORTBEN ASORT(BENEFICIARIES,,,"X","Y","X[1]<Y[1]")
```

In the above example, the SORTBEN array will contain all the elements of the BENEFICIARIES array, starting at the first element, where the first element is less than the second element.

For general information about arrays, see *Using Arrays* on page 147.

Sorting on Multiple Columns

In the <compare_expression> parameter, you need to include enough parameters (columns) to ensure that the results are sorted consistently.

For example, in one array, it may be sufficient to sort on one column, such as ID Number, whereas in another array, you may need to sort by Last Name and First Name.

The following example shows how to sort the array in ascending order, first by the first column, and then by the second column:

```
ASORT( aArray, , , "X", "Y", "(X[1] < Y[1]) .OR. ((X[1] =
Y[1]) .AND. (X[2] < Y[2]))")
```

where X and Y are the row names and 1 and 2 are the column numbers.

AT

Syntax: AT(<search_string>, <target_string>)

Parameter Name	Data Type	Description	Use
search_string	string	the string of characters to search for	required
target_string	string	the string on which to search for the search_string	required

Use either function to search for a particular character string and to calculate its position. The search is from left to right for AT and from right to left for RAT, but the value returned is calculated from the left relative to the string. If no match is found, a value of zero is produced. Using the AT (or RAT) function determines whether the particular character string exists, and what position it is in.

Both string parameters must be present, and both must be character strings or expressions that evaluate to character strings. The function searches string 2 for the first occurrence of string 1. If it finds string 1 in string 2, the function returns the position of the first occurrence of string 1 within string 2. If it doesn't find string 1 in string 2, the function returns zero.

Example:
```
DEFINE a "def"
DEFINE b "abcdefghijkldef"
```

```
DEFINE c AT(a,b)
```

result of c: 4

The variable `c` returns a value of 4 after the AT function is performed because the first occurrence of string `def` was found at position 4 in the string `abcdefghijkldef`.

Example:
```
DEFINE a "dfe"
DEFINE b "abcdefghijkldef"
DEFINE c AT(a,b)
```

result of c: 0

Variable `c` returns a value of 0 after the AT function is performed because the string `dfe` was not found in the string `abcdefghijkldef`.

Example:
```
DEFINE a "def"
DEFINE b "abcdefghijkldef"
DEFINE c RAT(a,b)
```

result of c: 13

Variable `c` returns a value of 13 after the RAT function is performed because the `c` of `def` is in position 13 numbering from left to right in the string `abcdefghijkldef`.

The following example uses the AT function to check a list of states to see if the `p_state` is listed.

Example:
```
IF AT(p_state,"AR,AK,KY,VA,CA,DE,ME,MQ")!=0
TEXT
this state is part of the list above
ENDTEXT
ENDIF
```

In the above example, if `p_state` is defined as `VA`, the string is found in string 2, providing a result greater than 0, and the text is printed in the document. If `p_state` is `WY`, it is not found in the target string, providing a result of 0, and the text is not included in the document.

CALL APP

Syntax: CALL APP(<program_ID>, <inputparameter>, <logicalparameter>)

Parameter Name	Data Type	Description	Use
<program_ID>	string	the program filename and path	required
<inputparameter>	string		optional
<logicalparameter>	string		required

Use this function to launch an external application. `program_ID` is a program identifier that indicates a program filename and path.

Assembler directly calls the external program indicated by `program_ID`. It pauses document generation until the launched external program terminates. It then looks for the file (`C:\`*`[IStream Document Manager folder]`*`\temp\defgen.tmp`) created by the external program containing the definition of its output parameters (name and value), which is treated by Assembler as its new internal variables. This file is in a Windows INI file format:

```
[DefineVariables]
defvar1=""value of defined variable 1""
defvar2=""value of defined variable 2""
defvar3=15
```

Each line in the `[DefineVariables]` section is used to define a variable. The left side of the equal (=) sign is the variable name. The right side is the value. Character values must have two quotation marks at each end. Numeric values should not have any quotation marks.

After all of the lines have been processed, the CALLAPP function deletes the `C:\`*`[IStream Document Manager folder]`*`\temp\defgen.tmp` file and assembly continues. The variables that were just defined can be referenced by subsequent expressions in the model document. If the external program does not create the `defgen.tmp` file, Assembler does not find this file and proceeds with generation. For example, the external program may simply update or create a user database or some file with a predefined name without passing back any specific parameters.

CASEPHRASE

Syntax: CASEPHRASE (<sentence>, <separator1>,<separator2>,...,<separatorN>)

Parameter Name	Data Type	Description	Use
sentence	string	the string to have all words capitalized	required
separator1	string	indicates the word that follows this separator is to be capitalized	optional
separator2	string	indicates the word that follows this separator is to be capitalized	optional
separatorN	string	indicates the word that follows this separator is to be capitalized	optional

This function is used to capitalize the first letter of every word (separated by `<separator1>...<separatorN>`) in `<sentence>`. The separator that is used defaults to a space if no separators are given.

Example:
```
argument: city
type: character
value: minneapolis_st.paul
CASEPHRASE(city,"-",".")
Generated Result: Minneapolis-St.Paul
```

Example:
```
argument: name
type: character
value: barb smith
CASEPHRASE(name)
Generated Result: Barb Smith
```

CASEWORD

Syntax: CASEWORD(<sentence>)

Parameter Name	Data Type	Description
sentence	string	the string containing the first word to be capitalized

Use this function to capitalize the first letter of the first word in `<sentence>`.

Example:
```
argument: p_holdnam
```

```
type: character
value: computer suppliers of america
CASEWORD(p_holdnam)
Generated Result: Computer suppliers of america
```

CDOW

Syntax: CDOW(<date>)

Parameter Name	Data Type	Description
date	date	the date you want generated as a character string

Use CDOW to produce the day of the week from a date value as a character string. The first letter is uppercase and the rest of the string is lowercase.

If IStream Document Manager does not find a date, it produces a null string ("").

Example:
```
argument: p_date
type: date,8
value: 01012007
CDOW(p_date)
Generated result: Wednesday
```

CHECK_DAY

Syntax: CHECK_DAY(<num>)

Parameter Name	Data Type	Description
num	float or integer	the date range which determines if day or days is displayed in the generated result

Use this function to produce the word “day” or “days”.

If the numeric value is equal to one, IStream Document Manager uses the singular, otherwise it uses the plural.

Example: In this example, the input argument can be a date value. Use the CTOD function to convert the character string to a date value first. Use the GETDAY function to obtain the numeric value.

```
DEFINE submitdate CTOD("04 10 2007")
DEFINE approvdate CTOD("04 20 2007")
DEFINE wait approvdate - submitdate
TEXT
Your waiting period is <GETDAY(wait )>
<CHECK_DAY(GETDAY(wait))>
ENDTEXT
```

Generated result: Your waiting period is 10 days.

CHECK_MTH

Syntax: CHECK_MTH(<num>)

Parameter Name	Data Type	Description
num	float or integer	the value which determines if “month” or “months” is displayed in the generated result

Use this function to produce the word month or months.

If the numeric value is equal to one, IStream Document Manager uses the singular, otherwise it uses the plural.

Example:
```
*** "wait" variable is greater than one
DEFINE wait 3
TEXT
The waiting period is <wait> <CHECK_MTH(wait)>.
ENDTEXT
*** "wait" variable is one
DEFINE wait 1
TEXT
The waiting period is <wait> <CHECK_MTH(wait)>.
ENDTEXT
```

Generate results:

The waiting period is 3 months.

The waiting period is 1 month.

CHECKUP

Syntax: CHECKUP (<value>, <tablename>, <columnName>, <db_InfoSourcename>)

Parameter Name	Data Type	Description	Use
value	any	the value to be searched for in the column in the table	required
tablename	string	the name of the table that is to be searched for in the value	required
columnName	string	the column that is to be searched for the value if omitted, it is assumed that the value is checked in the first column of the table	optional
db_InfoSourceName	string	the name of the InfoSource to be used in the search if omitted, then a predefined System InfoSource (storing IStream Document Manager lookup tables such as FORMATS.DBF and LANGTBL.DBF) is used as a default	optional

Use this function to search a table for a particular value. CHECKUP evaluates to True to indicate that a match was found, and to False to indicate that a match was not found.

If `db_InfoSourcename`, which is an optional argument, is indicated, it must be a string constant expression. CHECKUP searches for a particular value that is in any column of the table. The value parameter must also match it. That is, if the first column of a table is a string, then the value parameter must be enclosed in quotes. If the first column is numeric, then no quotes are needed.

Unless you are certain that the value is in the table, use CHECKUP before LOOKUP to prevent LOOKUP from indicating an error if it does not find a match. You can also use CHECKUP to reduce rules around include statements.

Note: For rules or functions, the `InfoSourcename` can be a string expression or string constant. If it is a string constant, the names must be within double quotes.

Example: You want to see if a particular agent exists based on the agent number. The database contains a table called `AGENTS` with a column called `AGENTNO`. Assume that the InfoSource referring to the database containing your lookup table is called `GroupLookup`.

Your section code looks like this:

```
DEFINE exists CHECKUP(12345,"AGENTS",
"AGENTNO","GroupLookup")
IF exists == .F.
TEXT
Agent Number 12345 doesn't exist.
ENDTEXT
ELSE
TEXT
Agent Number 12345 exists.
ENDTEXT
ENDIF
```

CHR

Syntax: CHR(number_code)

Use this function to produce the character corresponding to the ASCII number code or Unicode reference that you enter.

Example: You want to insert a é into your document using the ASCII number code. The number code for the character é is 233, so your code appears as follows:

```
TEXT
Vous avez une allocation pour m<CHR(233)>dicaments.
ENDTEXT
```

The phrase in the assembled document is:

Vous avez une allocation pour médicaments.

CMONTH

Syntax: CMONTH(<date>)

Parameter Name	Data Type	Description
date	date	the date value used to produce a month as a character string

Use CMONTH to produce the month from a date value as a character string. The first letter is uppercase and the rest of the string is lowercase.

If IStream Document Manager does not find a date, it produces a null string ("").

Example:

```
argument: p_date
type: date,8
value: 01012007
CMONTH(p_date)
Generated result: January
```

CTOD

Syntax: CTOD(<date>)

Parameter Name	Data Type	Description
date	string	the date value used to convert a character string consisting of numbers representing the month, day and year, separated by any character other than a number, into a date value

This function converts a character string of numbers representing the month, day and year, separated by any character other than a number, into a date value. You must specify the month, day and year digits in the format MM/DD/CCYY. If the string does not contain a valid date, it produces an empty date value. If the century digits are not specified, the century is determined based on the format MM/DD/CCYY, or it defaults to “19”. CTOD converts the character field type to a date field type to apply an MDY function, resulting in the long format of month, day, year.

Example:
```
fieldname: p_date
type: character,8
value: 06/21/2007
TEXT
MDY(CTOD(p_date))
ENDTEXT
```

Generated result: June 21,2007

Example:
```
DEFINE cdate "04-23-2007"
TEXT
<CTOD(cdate)>
ENDTEXT
```

Generated result: 04/23/2007

DATE

Syntax: DATE()

Use this function to retrieve the system date (current date on your computer) and return a value of MM/DD/YYYY.

Example: In this example, the DATE function is combined with `MDY` to obtain the system date and convert it to the stated month, day, year format. Note the comment in the example, marked by ***:

```
MDY(date())
```

Generated result: January, 1, 2007

```
*** Retrieves the system date and converts it to
```

```
*** a month, day, year format using MDY function.
```

Generated result: January 1, 2007

DAY

Syntax: DAY(<date>)

Parameter Name	Data Type	Description	Use
date	date	the date value used to extract the day of the month as a number between one and thirty-one	required

Use this function to extract the day of the month as a number between one and thirty-one. The input argument must be a date value, but not a constant. Use the CTOD function to convert the character string to a date value first.

If there is no date, the day value is zero.

Example:
```
argument: p_date
type: date, 8
value: 11012007
DAY(p_date)
Generated result: 01
```

DISPVAR

Syntax: CALL DISPVAR(<variable>)

Parameter Name	Data Type	Description
variable	string	the variable type to be displayed as a variable with the generation log

DISPVAR is a function you can use for debugging your document. It enables you to display the value of a defined variable with the generation log. You can display all variable types using this rule. The output is displayed in the compile/ generation log. For instructions on how to save or print the generation log, see *Saving the Compile/Generation Log* on page 169 and *Printing the Compile/ Generation Log* on page 169.

If you need to know the value of a field in the database, define a variable as the database field and then use DISPVAR.

Note: DISPVAR needs CALL SHOWRULES (.T.) activated to display variables.

Example:
```
DEFINE Life c_life
CALL DISPVAR("Life")
```

DLL

Syntax: DLL(<dllFileName>,<funcName>,<inputValue>)

Parameter Name	Data Type	Description	Use
dllFileName	string	the name of the DLL indicates the DLL filename and path	required
funcName	string	the name of the function in the DLL to be called	required
inputValue	string	a value to be passed to function	optional

This rule enables you to run an external Dynamically Linked Library (DLL) during generation. All arguments in this rule must be character strings or expressions that evaluate to character strings. When Assembler encounters a call to the DLL function in the model document, it calls the specified function and waits until the called function returns a string value.

As soon as the called DLL function terminates, Assembler looks for the file (`C:\`*`[IStream Document Manager folder]`*`\temp\defgen.tmp`). This file contains the definition of its output parameters (name and value), which is treated by Assembler as its new internal variables. After `defgen.tmp` file is read the DLL function deletes it. If Assembler does not find the `defgen.tmp` file (that is, the DLL function has not created it) it proceeds. The DLL remains in memory until Assembler finishes.

Example: `C:\System\Libraries\Financial.dll` contains a Rate function which calculates the rate for life insurance policy based on person's age, meaning it takes input value `Age` as its input parameter. Enter the IStream Author code for the model document section as follows:

```
DEFINE Age "57"
*** Alternatively Age can be a database field.
TEXT
This is the rate we propose for your Life insurance policy:
<DLL("C:\System\Libraries\Financial.dll","Rate",Age)>
ENDTEXT
```

DMY

Syntax: DMY(<date>)

Parameter Name	Data Type	Description
date	date or integer	the numeric date to convert into a DD-MM-YYYY date format

Use this function to convert a numeric date into a date in the format DD-MM-YYYY. You can also customize this function with the FORMATS table. The input format is CCYYMMDD or YYMMDD or CYYMMDD.

Example:
```
argument: mdate
type: numeric, 8
value: 20070630
<DMY(mdate)>
Generated result: 30-Jun-2007
```

Example:
```
*** date function is combined with CTOD to obtain
*** the stated day, month and year format.
DEFINE mdate7 CTOD("10 31 07")
TEXT
<DMY(mdate7)>
ENDTEXT
Generated result: 31-Oct-2007
```

DOCERR

Syntax: CALL DOCERR(<error_message>)

Parameter Name	Data Type	Description
error_message	string	the error message used to initiate a user-generated error or test a section

Use the DOCERR function when you want to initiate user-generated errors, or to test a section. The output appears in the compile/generation log.

As an example, you can use this function inside an IF statement to verify that policy data exists, or that it is within specified limits.

Example:
```
IF c_lifemax:limits > 1000000
CALL DOCERR("Policies cannot have more than 1 million life
coverage.")
ENDIF
```

If the Life Maximum is greater than $1 million dollars, you see the following when generating:

```
ERROR:Assembler Engine: User generated error: Policies
cannot have more than 1 million life coverage.
```

DOL_AMT

Syntax: DOL_AMT(<value>)

Parameter Name	Data Type	Description
value	integer or float	the numeric version of an amount to convert to a financial format

Use this function to convert a numeric version of an amount into a financial format. This format contains a dollar sign at the beginning of the number, commas separating the thousands, and no decimal places. It also rounds the value to the closest whole number.

Example:
```
argument: c_maxlife
type: numberic
value: 123456
DOL_AMT(c_maxlife)
Generated Result: $123,456
```

Note: If you need to support values larger than 999,999,999, contact your system administrator.

DOL_AMT2

Syntax: DOL_AMT2(<value>)

Parameter Name	Data Type	Description
value	integer or float	the numeric version of an amount you want converted into a financial format

Use this function to convert a numeric version of an amount into a financial format. This format contains a dollar sign at the beginning of the number, and commas separating the value in thousands. This function rounds the `value` to two decimal places.

Example:
```
argument: c_maxlife
type: numeric
value: 123456
DOL_AMT2(c_maxlfe)
Generated Result: $123,456.00
```

Note: If you need to support values larger than 999,999,999, contact your system administrator.

DOW

Syntax: DOW(<date>)

Parameter Name	Data Type	Description
date	date	the date value used to calculate the day of the week as a number between one (Sunday) and seven (Saturday).

Use this function to calculate the day of the week as a number between one (Sunday) and seven (Saturday).

If IStream Document Manager cannot find a date, the date value is zero.

Example:
```
argument: p_date
type: date, 8
value: 01012007
January 01, 2007

DOW(p_date)
Generated result: 2
```

***indicates Monday, the 2nd day of the week

DTOC

Syntax: DTOC(<date>)

Parameter Name	Data Type	Description
date	date	the date value used to produce a character string representing a date, using the current date format

Use this function to produce a character string representing a date, using the current date format. The default format is MM/DD/YY. The input argument must be a date value. Use the CTOD function to the convert character string to date value first.

If IStream Document Manager cannot find a date, it produces a string of spaces equal in length to the current date format.

Example:
```
argument: p_date
type: date, 8
value: 18062007

DTOC(p_date)
Generated result: 06/18/07
```

DTOS

Syntax: DTOS(<date>)

Parameter Name	Data Type	Description
date	date	the date value used to produce a character string eight characters long, in the format YYYYMMDD

Use this function to produce a character string eight characters long, in the format YYYYMMDD. The input argument must be a date value. Use the CTOD function to convert the character string to date value first.

If IStream Document Manager cannot find a date, it produces a string of eight spaces.

Example:
```
DEFINE mdate CTOD("10 31 2007")
TEXT
<DTOS(mdate)>
ENDTEXT
```

Generated result: 20071031

EMPTY and !EMPTY

Syntax: EMPTY(<value>)

Parameter Name	Data Type	Description
value	integer, float, string, date, or boolean	the expression used to determine if its value is empty

Use the EMPTY function to evaluate any type of expression to determine if the value is empty. If the value is empty, the expression is True, otherwise, the expression is False.

In this function:

- a character field is empty if no data exists (a space does not indicate that it is empty)
- a numeric field is empty if it contains 0
- a date field is empty if no date exists
- a logical field is empty if it is False

Use !EMPTY to check if a field is not empty.

Example:
```
IF EMPTY(PLAN_CODE)
TEXT
There is no plan code for this policy.
```

```
ENDTEXT
ENDIF
```

***This checks to see if there is any data in the field plan_code.

***If no data is found, the text statement is produced.

EXP

Syntax: EXP(<value>)

Parameter Name	Data Type	Description
value	float or integer	the float or integer expression used to calculate the antilog

Use this function to calculate the antilog (the inverse of the log).

Example:
```
argument: amount
type: numeric, 2
value: 1
EXP(amount)
Generated result: 2.718282
```

FIELD

Syntax: FIELD(<tablename>,< fieldname>)

Parameter Name	Data Type	Description	Use
tablename	string	the tablename that is to be used for the fieldname	required
fieldname	string	the fieldname that occurs in multiple tables	required

Use this function to refer to all field names that occur in multiple tables.

Example: The field `polno` exists in both the policy and class tables. You need to define which table should be used.

```
TEXT
The policy number is <FIELD("policy","polno")>
ENDTEXT
```

Note: The FIELD function can only be used with UserDatabase InfoSources. If you are working with other types of InfoSources and have fields with the same name from different tables, use the QUERY function and its ROWSET parameter.

FINDFILE

Syntax: FINDFILE(<itemreference>, <InfoSourcename>)

Parameter Name	Data Type	Description	Use
itemreference	string	the section's filename	required
InfoSourcename	string	a particular file system InfoSource name	required for documents stored in the DMS, optional for others

Use this function to determine whether a model document section exists before including it, or that a text file exists before using GETFILE.

If `InfoSourcename` represents a file system type InfoSource, then `itemreference` must be a file name. If `InfoSourcename` represents an IStream Document Manager type InfoSource, then `itemreference` must be the name and path of the section's document.

`InfoSourcename` is only required for documents stored in the DMS. If `InfoSourcename` is not used, the complete path to the document must be used.

Example: **With the InfoSourcename**

```
IF FINDFILE("Models\Policy\benefits.cds", "G_Ben")=.T.
INCLUDE G_Ben:"benefits.cds"
ENDIF
```

Without the InfoSourcename

```
IF FINDFILE("c:\Models\Main\Section1.cds") = .T.
Include Models:"Main\Section1.cds"
ENDIF
```

This function can also be used to specify a path (UNC supported) to a file stored in the file system using the following syntax:

Syntax: FINDFILE(<filename>)

Parameter Name	Data Type	Description
filename	string	the path to a file stored in the file system

GETDAY

Syntax: GETDAY(<date>)

Parameter Name	Data Type	Description
date	date	the date value used to produce a numeric value from 1 to 31, representing the day in the date

Use this function to produce a numeric value from 1 to 31, representing the day in the date. The input argument must be a date value. Use the CTOD function to convert a character string to date value first. If IStream Document Manager cannot find a date, the day value is 0.

Example:
```
argument: p_date
type: date,8
value: CTOD ("01/31/94")
GETDAY(p_date)
Generated result: 31
```

GETFILE

Syntax: GETFILE(<itemreference>,<InfoSourcename>)

Parameter Name	Data Type	Description	Use
itemreference	string	the filename for the text to be inserted into an assembled document	required
InfoSourcename	string	a particular file system InfoSource name where the `itemreference` is located	required

Use this function to insert text into the assembled document from a text file (TXT) instead of from a model document section. The file must be regular text (ASCII). Use it within a TEXT...ENDTEXT block, enclosing the GETFILE function in

angle brackets so it is evaluated and does not print as text. The contents of the text file are inserted in your document at the point of the <GETFILE> command.

Note: You cannot use the CALL rule in combination with the GETFILE function

Example:
```
IF FINDFILE("newrate.txt", "netfile")=.T.
TEXT
<GETFILE("newrate.txt", "netfile")>
ENDTEXT
ENDIF
```

Syntax: GETFILE(<filename>))

Parameter Name	Data Type
filename	string

Filename is the path (UNC supported) and name of the text document.

Example:
```
IF FINDFILE("C:\IStream\BOILERPLATE.TXT") = .T.
  TEXT
    <GETFILE("C:\IStream\BOILERPLATE.TXT")>
  ENDTEXT
ENDIF
```

GETIMAGE

Use this function to insert a valid image from an image file into an assembled document.

Place the `GETIMAGE` function within a `TEXT. . . ENDTEXT` block, enclosing the `GETIMAGE` function in angle brackets so it is correctly evaluated. The image will be inserted into the assembled document at the location of the `<GETIMAGE>` expression.

If the reference or the image type is invalid, a warning message is generated.

You can use `GetImage` with the following related functions:

- *ISIMAGE* on page 117
- *SETIMAGEFORMAT* on page 137

Image Function Limitations

The `GetImage`, `IsImage` and `SetImageFormat` functions have the following limitations:

- only .BMP and .JPG file formats are currently supported: other image types (including .TIF, .WMF and .PNG) are not supported
- PDFs are not supported because you cannot insert a PDF as an image into a Word document
- images from an IStreamXML or User DB InfoSource are not supported

GetImage Limitations

Please note the following limitations for the `GetImage` function:

- `GetImage` does not work inside Office Art objects such as text boxes.
- `GetImage` does not work inside a footnote or endnote section.
- You cannot use `GetImage` with a `CALL` statement, or in a `DEFINE` statement.

Note: If you are this function with hyperlinked parameters, such as hyperlinked UNC paths created by Microsoft Word, ensure that you have cleared all the Word hyperlink options as described in *Disabling Auto-Formatting in Word* on page 24.

Syntax

The `GetImage` function uses the following two syntaxes:

Syntax 1: GetImage(itemreference, InfoSourcename)

Note: This syntax currently supports only file system and IStream Document Manager InfoSources.

Parameter Name	Data Type	Description
itemreference	string	the file name of the image to be inserted into the assembled document
InfoSourcename	string	the name of the file system InfoSource or IStream Document Manager InfoSource where the `itemreference` is located Note: If this name is invalid (or the name of the InfoSource does not exist), an error message will appear both in the generation log and in the .CLG file.

Example:
```
DEFINE signed "john_smith.bmp"
IF ISIMAGE(signed, "esignature") = .T.
TEXT
   Sincerely,
   <GETIMAGE(signed, "esignature")>
   John Smith
ENDTEXT
ENDIF
```

Syntax 2: GetImage(refname)

This syntax currently supports images with local file paths (for example, `C:\IStream\image.jpg`) and UNC file paths.

Parameter Name	Data Type	Description
refname	string	the UNC-supported path and name of the image

Example:
```
DEFINE signed "C:\IStream\john_smith.bmp"
IF ISIMAGE(signed) = .T.
   TEXT
   Sincerely,
   <GETIMAGE(signed)>
   John Smith
   ENDTEXT
ENDIF
```

GETMONTH

Syntax: GETMONTH(<date>)

Parameter Name	Data Type	Description
date	date	the date value used to produce a number from 1 to 12, representing the month in the date

Use this function to produce a numeric value from 1 to 12, representing the month in the date. The input argument must be a date value. Use the CTOD function to convert character string to a date value first. If IStream Document Manager cannot find a date, the month value is 0.

Example:
```
argument: p_effdate
type: date,8
value: CTOD("01/31/07")=01/31/2007
GETMONTH(p_effdate)
Generated result: 1
TEXT
<GETMONTH(p_effdate)>
ENDTEXT
```

If the value of the variable is 01/31/2007, the number in the assembled document is 1.

GETYEAR

Syntax: GETYEAR(<date>)

Parameter Name	Data Type	Description
date	date	the date value used to produce a numeric value representing the year in the date

Use this function to produce a numeric value representing the year in the date. The input argument must be a date value. Use CTOD function to convert the character string to date value first. If IStream Document Manager cannot find a date, the YEAR value is 0.

Example:
```
argument: p_effdate
type: date,8
value: CTOD("01/31/07")=01/31/2007
GETYEAR(p_effdate)
```

Generated result: 2007

```
TEXT
<GETYEAR(p_effdate)>
ENDTEXT
```

If the value of the variable is 01/31/2007, the number in the assembled document is 2007.

IF

Syntax: <IF(<condition>, <true expression>, <false expression>)>

Parameter Name	Data Type	Description	Use
condition	string	the condition to check	required
true expression	string	the expression to generate if the IF rule evaluates to True	required
false expression	string	the expression to generate if the IF rule evaluates to False	required

You can use a variable IF statement within a text block to check a condition.

Important: The variable IF is enclosed in the angle (<>) brackets like a variable. It is also referred to as an *inline IF* statement. The syntax is the same as any other variable, that is, IF(...).

Note: The text expression has a maximum of 255 characters.

Use the Variable brackets, along with a rule and two expressions:

1. Enclose the expressions in quotation marks.
2. Type a comma after the rule.
3. Type a comma between the two expressions.

If the IF rule evaluates to True, IStream Document Manager prints the first expression in this variable in the assembled document.

If the IF rule evaluates to false, IStream Document Manager prints the second expression in this variable in the assembled document. IStream Document Manager prints whatever is between the double quotation marks; if there is nothing, then nothing prints.

Example:
```
argument: p_state
type: character, 2
value: CA
TEXT
This policy <IF(p_state="CA","does not include","includes")>
coverage for earthquakes.
ENDTEXT
Generated Result: This policy does not include coverage for
earthquakes.
```

INT

Syntax: INT(<numeric value>)

Parameter Name	Data Type	Description
numeric value	integer or float	the value to generate an integer portion of

Use this function to produce the integer portion of the value. It mathematically truncates the value.

Example:
```
argument: amount
type: numeric, 6
value: 123.45
INT(amount)
Generated result: 123
```

ISALPHA

Syntax: ISALPHA(<string_param>)

Parameter Name	Data Type	Description
string_param	string	the string used to determine whether an argument is made up of characters

Use this function to determine whether an argument is made up of characters.

If the first character of the string is alphabetic, this function is True. Otherwise, it is False.

Note: It only checks the first character.

Example:
```
argument: d_postal
type: character, 7
value: M5V 2R2
ISALPHA(d_postal)
Generated result: .T.
```

ISDIGIT

Syntax: ISDIGIT(<string_param>)

Parameter Name	Data Type	Description
string_param	string	the argument to check whether it contains numbers

Use this function to determine whether an argument contains numbers.

If the first character of the string is a number between 0 and 9, this function is True. Otherwise, it is False.

Note: It only checks the first digit.

Example:
```
argument: amount
type: character, 4
value: 0193
ISDIGIT(amount)
Generated result: .T.
```

ISFIELDNULL

Use the ISFIELDNULL function to evaluate a field to determine if it is null. If the field is null, the ISFIELDNULL function will return True. If the field is not null, it will return False.

When a value is returned from a database, it is checked for null constraints and the function will return a Boolean value. The function accepts one parameter which specifies a database field name to be checked for a null value. This function is similar to EMPTY and !EMPTY.

Example:
```
QUERY "SELECT column1 FROM table"
TEXT
IF ISFIELDNULL(column1)
*** Returns TRUE if column1 contains a null value
    ELSE
*** Returns FALSE if column1 does not contain a
    null value
ENDIF
ENDTEXT
```

ISIMAGE

Use this function to check if both a reference to an image file and the image file itself are valid. If they are both valid, the `ISIMAGE` function returns `True`. If either is invalid, `ISIMAGE` returns `False`.

Note: If you are this function with hyperlinked parameters, such as hyperlinked UNC paths created by Microsoft Word, ensure that you have cleared all the Word hyperlink options as described in *Disabling Auto-Formatting in Word* on page 24.

You can `IsImage` with the following related functions:

- *GETIMAGE* on page 111
- *SETIMAGEFORMAT* on page 137

Note: See *Image Function Limitations* on page 111 for the limtations on these image fucntions.

Syntax

The `IsImage` function has the following two syntaxes:

Syntax 1: IsImage(itemreference, InfoSourcename)

Note: This syntax supports only file system and IStream Document Manager InfoSources.

Parameter Name	Data Type	Description
itemreference	string	the file name of the image to be inserted into the assembled document
InfoSourcename	string	the name of the file system InfoSource or IStream Document Manager InfoSource where the `itemreference` is located

Example:
```
DEFINE signed "mysigned.bmp"
<ISIMAGE(signed, "esignature")>
```

Syntax 2: IsImage(refname)

This syntax currently supports images with local file paths (for example, `C:\IStream\image.jpg`) and UNC file paths.

Parameter Name	Data Type	Description
refname	string	the UNC-supported path and name of the image

Example:
```
DEFINE signed "C:\mysigned.bmp"
<ISIMAGE(signed)>
```

ISLOWER

Syntax: ISLOWER(<string_param>)

Parameter Name	Data Type	Description
string_param	string	the argument to check whether its left-most character is lowercase

Use this function to determine whether the left-most character in the argument is a lowercase letter. If it is, this function is True, otherwise, it is False.

Example:
```
argument: city
type: character, 10
value: new york
ISLOWER(city)
Generated result: .T.
***result equates to true because the left-most character is
***a lowercase letter.
```

ISUPPER

Syntax: ISUPPER(<string_param>)

Parameter Name	Data Type	Description
string_param	string	the argument to determine whether the left-most character is uppercase

Use this function to determine whether the left-most character in the argument is an uppercase letter. If it is, this function is True, otherwise, it is False.

Example:
```
argument: city
type: character, 10
value: new york
ISUPPER(city)
Generated result: .F.
```

LEFT

Syntax: LEFT(<name>, <index>)

Parameter Name	Data Type	Description	Use
name	string	the value used to produce a character string of a length specified by the index value	required
index	integer	the length for the resulting character string	required

Use this function to produce a character string of the length you specify, to the numeric count starting from the left of the string. You can use this to print only a portion of a database value.

If the string is shorter than the length you specify, IStream Document Manager produces the entire string. If the length you specify is a negative number or zero, IStream Document Manager produces a null string ("").

Example:
```
argument: dentcode
type: character, 12
value: 4048DENT8123
TEXT
The first two digits of the number on your Benefits card
indicates your dental plan type, for example <LEFT(dentcode,
2)>.
ENDTEXT
Generated result: The first two digits of the number on your
Benefits card indicates your dental plan type, for example
40.
```

LEN

Syntax: LEN(<string_param>) or LEN (<array_var>)

Parameter Name	Data Type	Description	Use
string_param	string	the character variable to determine the number of characters	required
array_var	array	the array to determine the size	required

Use this function to determine the number of characters in the specified variable. This can also be used to determine the number of lines in a two-dimensional array, or the number of elements in a one-dimensional array.

Example:
```
TEXT
<IF (LEN(Beneficiaries)= 1,"Beneficiary","Beneficiaries")>
ENDTEXT
```

In the above example, `Beneficiaries` identifies an array, but not a variable. If there is more than one line in the array, the word `Beneficiaries` will print, otherwise `Beneficiary` will print.

LOG

Syntax: LOG(<value>)

Parameter Name	Data Type	Description
value	integer or float	the value to calculate a natural logarithm with

Use this function to calculate the natural logarithm of the value in the variable.

If the value is less than or equal to zero, IStream Document Manager produces an error message in the generation log.

Example:
```
argument: amount
type: numeric, 8
value: 123.7654
LOG(amount)
Generated result: 4.818387838150676
```

LOOKUP

Syntax: LOOKUP (<value>,<tablename>, <column1>, <column2>, <InfoSourcename>)

Parameter Name	Data Type	Description	Use
value	variable name	the name of the variable from the current database	required
tablename	string	the tablename of the lookup table.	required
column1	string	the value to be found in the `column1` column and that is available in the current database if you do not specify a column, the first column of the table will be checked	optional, however if you specify a value, you must also specify values for the `column2` and `InfoSourcename` parameters.
column2	string	the column to look for a value in the lookup table and is the field that links to `column1` if you do not specify a column, the value will be retrieved from the second column if you specify a column, you must also specify values for the `column1` and `InfoSourcename` parameters	optional, however if you specify a value, you must also specify values for the `column1` and `InfoSourcename` parameters
InfoSourcename	string	the name of the InfoSource for the lookup table can be omitted if a predefined System InfoSource exists pointing to the system lookup tables to be used as the default.	optional, however if you specify a value, you must also specify values for the `column1` and `column2` parameters.

This function enables you to check and retrieve values from any table columns you specify. By default, the first column of a table is checked for values and the values are retrieved from the second column. However, you can override these defaults using the `column1` and `column2` parameters. For example, you could specify to check for the values in the third column and to retrieve the values from the fifth column.

This function enables you to check values not only in the first column and to retrieve values not only from the second column of a table, but also from any specified ones. The InfoSource can be an ODBC or IStreamXML type.

Note: For rules or functions, the `InfoSourcename` can be a string expression or string constant. If it is a string constant, the names must be within quotes.

Example: You have a database containing a lookup table called AGENTS. The table contains a column named AGENTNO and another named AGENTNAME. You want to print the name of a particular agent, based on the agent number. The InfoSource referring to the database containing your lookup table is called GroupLookup. You would author the section as follows:

```
DEFINE name LOOKUP (12345, "AGENTS", "AGENTNO", "AGENTNAME",
"GroupLookup")
TEXT
Agent Number 12345<name>.
ENDTEXT
```

LOWER

Syntax: LOWER(<string_param>)

Parameter Name	Data Type	Description
string_param	string	the character string to convert to lowercase

Use this function to convert all letters of the character string to lowercase.

Mainframe data often appears in uppercase letters. Use this function, along with CASEWORD or CASEPHRASE, to convert all letters to lowercase, and then to convert the first letters of every word or every sentence to uppercase.

Example:
```
before: BARB SMITH
LOWER(name)
after: barb smith
```

LOWTRIM

Syntax: LOWTRIM(<string_param>)

Parameter Name	Data Type	Description
string_param	string	the character string to convert to lowercase, removing leading and trailing spaces

Use this function to convert all characters to lowercase and remove leading and trailing spaces.

Example:
```
argument: occup
type: character, 25
value: Gas Technologist
LOWTRIM(occup)
Generated result: gas technologist
```

```
TEXT
Policy holder is employed as a <LOWTRIM(occup)>.
ENDTEXT
Generated result: Policy holder is employed as a gas
technologist.
```

LTRIM

Syntax: LTRIM(<string_param>)

Parameter Name	Data Type	Description
string_param	string	the character string to remove leading spaces from

Use this function to remove leading spaces in a character string.

Example:
```
argument: p_holdnam
type: character, 25
value: " Jane Smith"
LTRIM(p_holdnam)
```

Generated result: Jane Smith

```
TEXT
The Policyholder is <LTRIM(p_holdnam)>.
ENDTEXT
```

MAX

Syntax: MAX(<expression1>, <expression2>)

Parameter Name	Data Type	Description	Use
expression1	integer, float, or date	the expression with the lower amount	required
expression2	integer, float, or date	the expression with the larger amount	required

Use this function to select the larger of two expressions, which can be both numeric or both date.

Note: The data types must match. For example, `expression1` cannot be a date and `expression2` an integer. They must *both* be a date or an integer.

Example:
```
argument: premium1 and premium2
type: numeric (or date)
value: premium1=100, premium2=200
MAX(premium1, premium2)
```

```
Generated result: This is the greater amount of your two
premiums:200.
TEXT
This is the greater amount of your two premiums:
<MAX(premium1,premium2)>.
ENDTEXT
```

MDY

Syntax: MDY(<date>)

Parameter Name	Data Type	Description
date	date	the date to convert to Month DD,YYYY format

Use this function to convert a character, numeric, or date argument into a date in the format Month DD,YYYY. You can also customize this function with the FORMATS table.

Example:
```
argument: p_date
type: date, 8
value: 20070131
MDY(p_date)
Generated result: January 31, 2007.
***This function can also be used with the system date
TEXT
<MDY(date())>
ENDTEXT
***takes system date and converts it to Month DD, YYYY
format.
```

MIN

Syntax: MIN(<expression1>, <expression2>)

Parameter Name	Data Type	Description	Use
expression1	integer, float, or date	the expression with the lower value	required
expression2	integer, float, or date	the expression with the higher value	required

Use this function to select the smaller of two expressions, which can be both numeric or both date.

Note: The data types must match. For example, `expression1` cannot be a date and `expression2` an integer. They must *both* be a date or an integer.

Example:
```
argument: grpsize1 and grpsize2
type: numeric, 5
value: grpsize1=100, grpsize2=200
MIN(grpsize1,grpsize2)
Generated result: 100
TEXT
This premium is calculated based on the smallest group:
<min(grpsize1,grpsize2)>.
ENDTEXT
Generated result: This premium is calculated based on the
smallest group: 100.
```

MONTH

Syntax: MONTH(<date>)

Parameter Name	Data Type	Description
date	date	the date value used to produce a number from 1 to 12, representing the month in the date

This function is the same as the GETMONTH function, described in *GETMONTH* on page 113.

MYSTR1

Syntax: MYSTR1(<value>)

Parameter Name	Data Type	Description
value	integer or float	the numeric value to remove all leading and trailing spaces or zeroes from

Use this function to remove all leading and trailing spaces or zeroes in a numeric value. The maximum length of the numeric expression is 15 digits.

Example:
```
argument: amount
type: numeric, 9
value: 0000001.0
MYSTR1(amount)
Generated result: 1
```

MYUSERID

Syntax: MYUSERID()

Use this function to obtain the name of the user generating a document.

Note: This function does not work properly when you use web generation from the DMS to create your IStream document. The function returns a cookie name instead of the name of the user performing the generation.

Example:
```
TEXT
<MYUSERID()>
ENDTEXT
```

NUMBER

Syntax: Number(<value>)

Parameter Name	Data Type	Description
value	integer or float	the value to convert to a financial format

This function is very similar to DOL_AMT. Use this function to convert a numeric version of an amount into a financial format. This format contains commas but no dollar sign, rounds to the closest whole number, and removes any decimal values.

Note: If you need to support values larger than 999,999,999, contact your system administrator.

Example:
```
argument: amountin
type: numeric, 9
value: 123456.42
<NUMBER(amountin)>
Generated result: 123,456
```

NUMBER2

Syntax: NUMBER2(<value>)

Parameter Name	Data Type	Description
value	integer or float	the value to convert to a financial format

This function is similar to DOL_AMT2. Use it to convert a numeric version of an amount into a financial format. This format contains commas and two decimal places but no dollar sign, and rounds to two decimal places.

Example:
```
argument: amountin
type: numeric, 6
value: 123456.42
<NUMBER(amountin)>
Generated result: 123,456.42
```

Note: If you need to support values larger than 999,999,999, contact your system administrator.

PADC

Syntax: PADC(<string expression>, <numeric_length>, <character>)

Parameter Name	Data Type	Description	Use
string expression	string	the expression to pad	required
numeric_length	integer or float	the length of the final expression	required
character	string	the character with which to pad the expression	required

This function pads a result using the value of the expression as the center. Use it to generate a string of at least the length you specify in the numeric length variable, containing the result of the character, number, or date expression.

If the character representation of the result is not as long as the length you specified, IStream Document Manager adds padding characters to both the right and left to make up the difference. If you do not specify a padding character, the default character is a space. The padding characters must be enclosed in double quotation marks.

Example:
```
argument: premcode
type: character, 4
value: dent
PADC(premcode, 10, "a")
TEXT
The premium code is <PADC(premcode,10,"a")>.
ENDTEXT
Generated result: The premium code is aaadentaaa.
argument: class
type: character
value: 523
PADC(class,6,"0")
TEXT
The six-letter class number is <PADC(class,6,"0")>.
ENDTEXT
Generated result: The six-letter class number is 005230.
```

PADL

Syntax: PADL(<string_expression>, <numeric_length>, <character>)

Parameter Name	Data Type	Description	Use
string_expression	string	the expression to pad	required
numeric_length	integer or float	the length of the final expression	required
character	string	the character with which to pad the expression	required

This function pads a result to the left of the value of the expression. Use it to generate a string of at least the length you specify in the numeric length variable, containing the result of the character, numeric, or date expression.

If the character representation of the result is not as long as the length you specified, IStream Document Manager adds padding characters to the left to make up the difference. If you do not specify a padding character, the default character is a space. The padding characters must be enclosed in double quotation marks.

Example:

```
argument: premcode
type: character, 4
value: dent
PADL(premcode,10,"a")
TEXT
The premium code is <PADL(premcode,10,"a")>
ENDTEXT
Generated result: The premium code is aaaaaadent.
argument: class
type: numeric
value: 523
PADL(class,6,"0")
TEXT
The six-letter class number is <PADL(class,6,"0")>.
ENDTEXT
Generated result: The six-letter class number is 000523.
```

PADR

Syntax: PADR(<string_expression>, <numeric_length>, <character>)

Parameter Name	Data Type	Description	Use
string expression	string	the expression to pad	required
numeric length	integer or float	the length of the final expression	required
character	string	the character with which to pad the expression	required

This function pads a result to the right of the value of the expression. Use it to generate a string of at least the length you specify in the numeric length variable, containing the result of the character, numeric, or date expression.

If the character representation of the result is not as long as the length you specified, IStream Document Manager adds padding characters to the right to make up the difference. If you do not specify a padding character, the default character is a space. The padding characters must be enclosed in double quotation marks.

Example:

```
argument: premcode
type: character, 4
value: dent
PADR(premcode,10,"a")
TEXT
The premium code is <PADR(premcode,10,"a")>
ENDTEXT
```

Generated result: The premium code is dentaaaaaa.

```
argument: signdate
type: numeric
value: 09/09/2007
PADR(signdate,14)
TEXT
Signed on<PADR(signdate,14)> at the offices of Pletsch and
Smith.
ENDTEXT
***note that the padding character is a space - the default.
Generated result: Signed on 09/09/2007  at the offices of
Pletsch and Smith.
```

POWER

Syntax: POWER(<x,y>)

Parameter Name	Data Type	Description
x,y	integer or float	the values to be used for the exponential function

This function contains this mathematical notation: x^y, where both x and y can be real and/or integer numbers. The restriction is that x can not be negative if y is not an integer number. This is because the result may be a complex number, which is not supported. In terms of the IStream authoring language, this mathematical function is expressed as an exponential function:

power(x,y)

Here both x and y can be a constant, variable, or an expression. If the value of x is negative and y is not an integer value, a generation or compilation error occurs.

Example: `P = C (1 + r) t`

where

P = future value

C = initial deposit

r = interest rate (expressed as a fraction: e.g, 0.06)

t = number of years invested

```
DEFINE C 100
DEFINE r 0.015
DEFINE t 2
DEFINE P C*POWER((1+r),t)
TEXT
Compound Interest of 1.5% on a deposit of $100 for 2 years
is $<P>
ENDTEXT
```

PROMPT

Syntax: PROMPT(<windowtitle>,<message_text>,<default_prompt>, <maximum_length>)

Parameter Name	Data Type	Description	Use
windowtitle	string	the title of the pop-up window	required
message_text	string	the message or description that appears above the data entry box used to include a descriptive phrase or instructions for the user inputting the data	required
default_prompt	string	a default value that the user can change, or can accept by clicking OK the default and the value entered can be a character value only for the default parameter (third parameter), if no value is specified (that is, an empty string is included in the rule) and there is a previously saved value, it is entered as the default value; the user still has the option to accept or reject that default value	required
maximum_length	number	the maximum number of characters a user can enter	required

This function prompts for data during the generation process. Use it when you need data that is not stored in the database or that is not appropriate to store in the database.

Important: Make sure that you only use PROMPT when document generation is interactive. PROMPTS that appear during document assembly on Publisher workers can cause problems.

Note: To exclude one or more of the parameters, enter an empty string of two quotes ("").

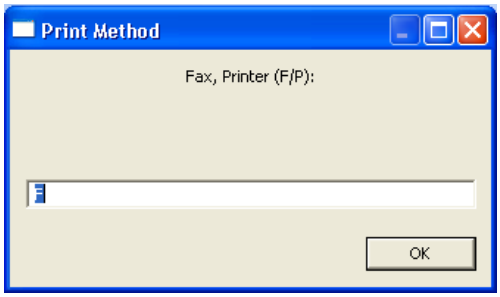
The PROMPT function's parameters determine the title of the dialog box, describe the data being prompted for, and include a default value that may be accepted or rejected at generation time. The final parameter is the maximum input value you set for the data. If you have a blank that, for example, requires the input to be a maximum of 30 characters long, set the final parameter to 30. Ensure to include commas between the parameters.

When Assembler encounters the PROMPT function, it displays a dialog box that requires specific information. Only character values are accepted. You can set a default value that can either be accepted or changed. However, if the user clicks **Cancel** to close the prompt window rather than inputting a value, a warning displays during generation.

Example:
```
DEFINE TESTVALUE PROMPT("Print Method","Fax, Printer (F/
P):","F",1)
```

Note: All four of the parameters must be entered. Be sure to include the commas.

The generated result is:



In this example:

- `windowtitle` is **Print Method**
- message_text is **Fax, Printer (F/P)**
- the `default_prompt` is **F**

Because the required input in the example below is a single letter (F or P), the input value has a maximum length of 1.

RAT

Syntax: RAT(<search_string>, <target_string>)

Parameter Name	Data Type	Description	Use
search_string	string	the string of characters to search for	required
target_string	string	the string to search for the `search_string`	required

This function finds the existence or position of a string within a target character string. Both parameters must be character strings or evaluate to character strings. RAT is different from AT in that it determines the position from the left relative to the string, but finds the first occurrence by searching right to left. See *AT* on page 93.

Example:
```
DEFINE a "def"
DEFINE b "abcdefghijkldef"
DEFINE c RAT(a,b)
result of c: 13
```

Variable `c` returns a value of 13 after the RAT function is performed because the `d` of `def` is in position 13 numbering from left to right in the string `abcdefghijkldef`.

REPLACESTR

Syntax: REPLACESTR(<source_string>, <old_string>, <new_string>)

Parameter Name	Data Type	Description	Use
source_string	string	the source string containing the string to replace	required
old_string	string	the string to be replaced	required
new_string	string	the string that will replace the old string	required

Use this function to replace strings. Each old string will be replaced with the new string in each instance of the indicated source string. The source string can therefore grow or shrink depending on the length of the old and new strings.

Note: This function is useful for avoiding apostrophe errors. If you replace a single apostrophe with a double apostrophe, you cause the first apostrophe to be recognized as an escape sequence, and the second apostrophe is considered part of the string.

Example:
```
argument: new string
type: character
value: "football"
REPLACESTR("Everybody likes ice hockey","hockey",
           "football")
Generated Result:"Everybody likes ice football"
```

Example:
```
argument: new string
type: character
value: "
REPLACESTR("O'Leary","'",""")
Generated result: "O"Leary"
```

Note: The above two examples indicate in simple terms the logic of this function. The next example provides a realistic application related to the second example above.

Example:
```
Define Key1 "O'Leary"
QUERY "SELECT * FROM TABLE1 WHERE COLUMN1 = "+""+
REPLACESTR(Key1,"",""")+"'","Datasource"
```

REPLICATE

Syntax: REPLICATE(<repeat_string>, <num_copies>)

Parameter Name	Data Type	Description	Use
repeat_string	string	the string to copy	required
num_copies	integer	the number of times to copy the string	required

Use this function to produce a character string containing the number of copies of the string that you specify, up to a maximum of 65,535 (64K) bytes in length. Unlike the SPACE function, REPLICATE requires two parameters.

If you specify a count of zero, IStream Document Manager produces a null string ("").

Example:
```
argument: amount
type: character, 4
value: 1234
REPLICATE(amount,3)
Generated result: 123412341234
```

RIGHT

Syntax: RIGHT(<orig_string>, <num_chars>)

Parameter Name	Data Type	Description	Use
orig_string	string	the original string to produce another, shorter string from, and that is `num_chars` long	required
num_chars	integer	the length for the produced character string	required

Use this function to return a given number of characters from the right of a given string.

If the string is shorter than the length you specify, IStream Document Manager produces the entire string. If the length you specify is a negative number or zero, IStream Document Manager produces a null string ("").

Example:
```
argument: dentcode
type: character, 12
value: DENT40498123
TEXT
The last three digits of the number on your benefits card
indicates your dental plan type, for example
<RIGHT(dentcode, 3)>.
```

```
ENDTEXT
Generated result: The last three digits of the number on
your benefits card indicates your dental plan type, for
example 123.
```

ROUND

Syntax: ROUND(<value>, <num_decimals>)

Parameter Name	Data Type	Description	Use
value	string	the value to round down	required
num_decimals	integer	the number of decimals to round the value down to	required

Use this function to calculate the numeric value with the number of decimals closest to the number you specify.

If you specify a negative number of decimals, IStream Document Manager produces an error.

Examples of the ROUND function are:

Code	Result
ROUND(50.00000,2)	50.00
ROUND(50.89798,2)	50.90
ROUND(50.0,2)	50.00
ROUND(50.002,0)	50
ROUND(50.898,-2)	error

Example:

```
argument: amount
type: numeric, 9
value: 123.24765
ROUND(amount,2)
```

Generated result: 123.25

RTRIM

Syntax: RTRIM(<trim_string>)

Parameter Name	Data Type	Description
trim_string	string	the string to remove trailing spaces from the right

Use this function to remove trailing spaces in a character string.

Example:
```
DEFINE A "            Downey         "
DEFINE B "          Herbert        "
TEXT
The policy holder is (<A>), (<RTRIM(B)>).
ENDTEXT
```

Generated result:

```
The policy holder is (          Downey           ), (
Herbert).
```

SETFORMAT

Syntax: CALL SETFORMAT (<formatname>)

Parameter Name	Data Type	Description
formatname	string	the language to format dates and monetary amounts

Normally, IStream Document Manager converts date and monetary functions to English format, regardless of the language of the document. Using CALL SETFORMAT allows you to represent values derived through functions according to the standards of a language other than English. For example, decimal points (.) are often represented as commas (,) in French. If you were creating a French document, you might use CALL SETFORMAT to make sure that numbers are represented the way that you want in generated IStream documents.

Example:
```
IF p_lang = "F"
CALL SETFORMAT("french")
CALL SETLANGUAGE("french")
ELSE
CALL SETFORMAT("english")
CALL SETLANGUAGE("english")
ENDIF
```

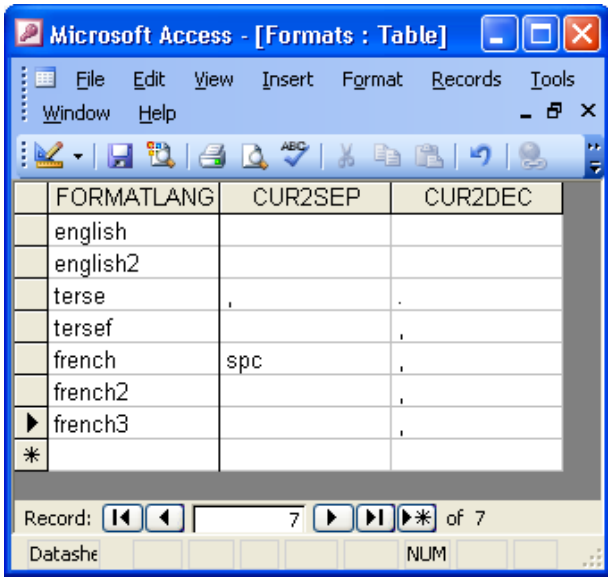
Related Topic

- *Customizing Decimal Separators* on page 137

Customizing Decimal Separators

The type of decimal and thousands separators used for each language format depends on the settings in the `FORMATS` table in `calsystem.mdb`. You can customize which separators are used for each language format by opening and editing the `FORMATS` table. Usually, `calsystem.mdb` can be found in the following location:

`[IStream Document Manager install folder]``\Components\Shared`



The **CUR2SEP** field determines what is used for the thousands separator, while the **CUR2DEC** field determines what is used for the decimal separator.

You cannot enter white spaces into the table. If you want to use a white space as a separator, enter **spc** instead.

If you leave a field blank, the default values are used as separators. For the thousands separator (**CUR2SEP**), the default is no thousands separator. For the decimal separator (**CUR2DEC**), the default value is a period (.) .

SETIMAGEFORMAT

Use this function to change the default image properties within a current generation session. If an invalid property name or value is specified, the default value does not change and a warning message is generated.

You can use `SetImageFormat` with the following related functions:

- *GETIMAGE* on page 111
- *ISIMAGE* on page 117

Note: See *Image Function Limitations* on page 111 for the limtations on these image fucntions.

Syntax: CALL SETIMAGEFORMAT(propertyname, propertyvalue)

Parameter Name	Data Type	Description
propertyname	string	the name of the property: layout, height, width or unit
propertyvalue	numeric	a valid value for the `propertyname`

Note: You cannot use different `SETIMAGEFORMAT` settings within a single `TEXT/ENDTEXT` block. Instead, you need to separate the block into smaller blocks. You can then include a different `CALL SETIMAGEFORMAT` statement before each `TEXT/ENDTEXT` block.

Image Properties

The following table lists the valid image property names and values. 0 is the default value for all properties.

Property	Values
layout	• 0 – insert graphic in line with text: this is the only valid value
height	• 0 – use the original image height (this is the default value) • any positive number: the target height as measured in `Units`
width	• 0 – use the original image width (this is the default value) • any positive number: the target width as measured in `Units`
unit	the unit of measurement for the `height` and `width`: • 0 – inches (this is the default value) • 1 – centimeters • 2 – millimeters • 3 – points (72 points = 1 inch) • 4 – picas (6 picas = 1 inch)
LockAspectRatio	controls whether or not the image aspect ratio (proportions) are locked • 0 – turns off aspect ratio locking • 1 – locks the image aspect ratio by width • 2 – locks the image aspect ratio by height

Note: Property names are not case-sensitive.

Example: The following code changes the image size to 3.5 cm x 2.3 cm, and locks the image aspect ratio so that a change in image width will be reflected by a proportional change in image height:

```
CALL SetImageFormat("unit", 1)
CALL SetImageFormat("width", 3.5)
```

```
CALL SetImageFormat("height", 2.3)
CALL SetImageFormat("LockAspectRatio", 1)
```

SETLANGUAGE

Syntax: CALL SETLANGUAGE(<language>)

Parameter Name	Data Type	Description
"language"	string	the language used to read or change the value of the language variable

Use this function to read or change the value of the language variable, which defaults to English. IStream Document Manager uses the letter corresponding to the current language value to select which TEXT blocks to include in the assembled document. The table which associates language names with letters for the TEXT rule is `LANGTABL`. It is accessed during assembly through a predefined System type InfoSource. It is located in the following Microsoft Access database: `\[`*`IStream Document Manager folder`*`]\Components\Shared\CalSystem.mdb`

Note: This rule does not work with an Oracle database.

If you do not specify a language character, IStream Document Manager uses the current language setting. For more information, see *TEXT with Language Settings* on page 84.

Example:
```
IF UPTRIM(p_language) == "E"
CALL SETLANGUAGE("English")
ENDIF
IF UPTRIM(p_language) == "F"
CALL SETLANGUAGE("French")
ENDIF
```

SHOWRULES

Syntax: CALL SHOWRULES(<bool_value>)

Parameter Name	Data Type	Description
bool_value	boolean	the boolean value that will trigger the function

Use this function to expand the messages and actions written to the generation log (in Author, Assembler, Microsoft Word or Adobe Acrobat) while the model document is being assembled. You can then refer to and print the generation log.

Insert SHOWRULES (.T.) at the beginning of the section you want to check. To stop extended messaging to the generation log at the end of the section, insert

SHOWRULES (.F.). To continue the messaging to the end of that section, do not insert SHOWRULES (.F.).

Note: The generation log will not display the detailed output when the SHOWRULES function is enabled. To view this detailed output, you need to select **View Generation Log**.

Important: This function is for testing only, so ensure that you remove it before moving your section into production.

Example:
```
CALL SHOWRULES(.T.)
```

SPACE

Syntax: SPACE(<numeric>)

Parameter Name	Data Type	Description
numeric	number	the number of spaces the number field will contain

This function is similar to the REPLICATE function. Where REPLICATE repeats a string any number of times you define, SPACE produces the number of spaces you specify to represent a number field.

Example:
```
TEXT
Dated this <SPACE(5)>th day of <SPACE(12)>, 19<SPACE(5)>.
ENDTEXT
Generated Result: Dated this   th day of          , 19     .
```

SPELLNUMBER

Syntax SpellNumber(refname)

Parameter Name	Data Type	Description
refname	numeric	a constant, variable or InfoSource field

This function spells out a number into its equivalent English wording.

Note:

- if `refname` is negative, the word “Minus” will appear at the beginning of the number
- all decimals are ignored

Examples

The following table illustrates the results of using the `SpellNumber` function with various numbers:

Example	Result
`SpellNumber(102)`	One Hundred and Two
`SpellNumber(23963)`	Twenty Three Thousand Nine Hundred and Sixty Three
`SpellNumber(0)`	Zero
`SpellNumber(553.9)`	Five Hundred and Fifty Three
`SpellNumber(-304)`	Minus Three Hundred and Four

SQRT

Syntax: SQRT(<value>)

Parameter Name	Data Type	Description
value	integer or float	the value on which to calculate the square root

Use this function to calculate the square root of a value.

If the value is negative, IStream Document Manager produces the value zero.

Example:
```
argument: amount
type: numeric, 2
value: 16
SQRT(amount)
Generated result: 4
```

STR

Syntax: STR(<value>, <length>, <num_decimals>)

Parameter Name	Data Type	Description	Use
value	integer or float	the numeric value to convert to a string.	required
length	integer	the length of the output string	optional
num_decimals	integer	the number of decimal places for the output string value	optional

Use this function to convert the numeric expression that you specify to a character string.

Note: Specifying a length larger than the length of the value is equivalent to using the length of the value.

Example:
```
argument: age
type: numeric, 2
value: 55
TEXT{}
{IF (AGE < 65, "Employees are covered to the earlier of age"
+ STR(AGE) + "or retirement", "Employees are covered to age
65.")}
ENDTEXT
```

If the age is 55, the wording in the assembled document is:

Employees are covered to the earlier of age 55 or retirement.

SUBSTR

Syntax: SUBSTR(<base_string>, <start_location>, <num_chars>)

Parameter Name	Data Type	Description	Use
base_string	string	the string to extract a substring from	required
start_location	integer	the starting position to begin the substring in	required
num_chars	integer	the number of characters to extract from the base string	required

Use this function to extract characters from a character string. Specify the starting position in the character string with the numeric start variable, and the number of characters to extract in the numeric count variable.

If the start value is positive, IStream Document Manager starts to count from the left-most character. If the start value is negative, IStream Document Manager starts to count from the right-most character.

Along with the RIGHT and LEFT functions, this function is useful for extracting portions of a character string.

Example:
```
argument: phone
type: character, 10
value: 9055131400
TEXT
To reach our offices, call (<SUBSTR(phone,1,3)>)
<SUBSTR(phone,4,3)>-<SUBSTR(phone,7,4)> during business
hours.
ENDTEXT
```

If your phone numbers are stored in a character field in the format 9055131400, the phone numbers in the assembled document appear in the format: (905) 513-1400.

```
Generated result: To reach our offices, call (905) 513-1400
during business hours.
```

TRIM

Syntax: TRIM(<trim_string>)

Parameter Name	Data Type	Description
trim_string	string	the value to remove all trailing spaces from

Use this function to remove all trailing spaces in a character string.

Example:
```
argument: pol_holder
type: character, 25
value: "Alain Michaud  "
TEXT
The policy holder is <TRIM(pol_holder)>.
ENDTEXT
Generated result: The policy holder is Alain Michaud.
```

TRUNCATE

Syntax: TRUNCATE(<value>)

Parameter Name	Data Type	Description
value	float	the value to truncate the decimal point and trailing zeroes from

Use this function to remove the decimal point and trailing zeroes from a value. The function searches for zeroes to the right of the decimal and truncates them.

Note: This function does not perform a mathematical truncation, but simply trims the zeroes.

Example: In this example, the value of the variable is displayed without the decimal and trailing zeros. In the database, the value of the variable is 500.00.

```
argument: value
type: numeric, 6
value: 500.00
TEXT
All claims greater than $<TRUNCATE(value)> should be
reported to head office.
ENDTEXT
```

If the value of the variable is 500.00, the number in the assembled document will be 500.

If the value of the variable is 500.010, the number in the assembled document will be 500.01.

```
Generated result: All claims greater than $500 should be
reported to head office.
```

UPPER

Syntax: UPPER(<string_param>)

Parameter Name	Data Type	Description
string_param	string	the character string to convert to uppercase

Use this function to convert the alphabetic characters in the character string to uppercase.

Example:

```
argument: p_holdnam
type: character, 25
value: Computer Suppliers of America
TEXT
This policy is issued in the name of <UPPER(p_holdnam)>.
```

```
ENDTEXT
Generated result: This policy is issued in the name of
COMPUTER SUPPLIERS OF AMERICA.
```

Note: Accented characters are not converted to uppercase. For example, QUÉBEC would appear as QUéBEC.

UPTRIM

Syntax: UPTRIM(<string_param>)

Parameter Name	Data Type	Description
string_param	string	the string to convert to uppercase and remove all leading and trailing spaces

Use this function to convert characters to uppercase and remove all leading and trailing spaces in a character string.

Example:
```
argument: name
type: character, 25
value: Alejandro Rojas
TEXT
This policy is issued to UPTRIM(name).
Generated result: This policy is issued to ALEJANDRO ROJAS.
```

VAL

Syntax: VAL(<string_number>)

Parameter Name	Data Type	Description
character string	string	the character string to convert to a numeric value and remove leading zeros

Use this function to convert the character string to a numeric value and drop leading zeros.

Example:
```
argument: cPolnum
type: character, 12
value: 0000012345
TEXT
The policy number is <VAL(cPolnum)>.
ENDTEXT
Generated result: The policy number is 12345.
```

YEAR

Syntax: YEAR(<date>)

Parameter Name	Data Type	Description
date	date	the value to extract the year from

Use this function to calculate the year from the date. The input argument must be a date value. Use CTOD function to convert character string to date value first. If IStream Document Manager cannot find a date, the year value is zero.

Example:
```
argument: effdate
type: date, 8
value: 20070131
TEXT
Your policy is effective starting the policy year
<YEAR(effdate)>.
ENDTEXT
Generated result: Your policy is effective starting the
policy year 2007.
```

YMD

Syntax: YMD(<date>)

Parameter Name	Data Type	Description
date	date	the to convert to a date in the format YYYY/MM/DD.

Use this function to convert a numeric argument or a date argument into a date in the format YYYY/MM/DD.

You can also customize this function with the FORMATS table.

Example:
```
argument: effdate
type: date
value: 20070131
TEXT
Your policy is effective as of <YMD(effdate)>.
ENDTEXT
Generated result: Your policy is effective as of 2007,
January 31.
```

Using Arrays

Arrays are groupings of data that you can access to populate an assembled document. You can also use an array to populate data into a chart or graph you have inserted into a model document or section: see *Inserting Charts and Graphs* on page 213. An array is a virtual table with columns and rows. The data in the array is referenced by numbers rather than by field names. For example, you could build an array of the beneficiaries for a policy:

Beneficiaries

Doe, Jane	Spouse
Doe, Fred	Son
Doe, John	Son
Andrews, Mary	Daughter

If you use a specific set of data a number of times in an assembled document, it is faster to access the data in an array rather than directly from the database, because the data in an array is in memory during generation. Arrays are particularly helpful if you need to access multiple tables in your database to retrieve the information that is repeated in your document.

Each piece of data in an array is referred to as an *element*. When using the data in a rule or variable insertion, you must refer to the element specifically. If the array above is called `Beneficiaries` and you wanted to print the name of the first beneficiary in the array, you would enter:

```
<Beneficiaries[1,1]>
```

The first number in the square brackets refers to the line or row [R] in the array. The second number refers to the number of items or columns [C] across. For the array to work, a value of at least 1 must be entered for either number.

Method: Using and manipulating arrays

To use an array, you need to:

1. Define the array: see *Creating an Array* on page 148.
2. Add data to the array using the CALL AADD function

To manipulate an array, you can:

- list all items in an array using the DO WHILE rule: see *DO WHILE on page 60*
- determine the number of rows in an array using the LEN function: see *LEN* on page 120
- sort data in an array using the ASORT function: see *ASORT* on page 92

Creating an Array

This section describes how to create an array.

Syntax: DEFINE arrayname {}

`Arrayname` is a valid variable name assigned to be the name of the array followed by two brackets "{}". These two parameters are required in defining an array.

Example

The following data is in the Class table:

C_polno	C_classdesc	Classcode
123451	Executive	123
123451	Clerical	212
123451	Maintenance	333

The following code shows how to define the array:

```
DEFINE theclasses {}
***Define the key data variable "nClassnumber"
DEFINE nClassnumber 1
***Read the data from the table, there are 3 data entries in
this example.
QUERY SELECT * from class", "tempdb"
DEFINE cCode ""
IF classcode =="123" .or. classcode=="333"
DEFINE cCode "flat"

ELSE
DEFINE cCode "multiple"
ENDIF
***Add the data to the array
CALL AADD (theclasses,{nClassnumber,c_classdesc,cCode})
DEFINE nClassnumber nClassnumber + 1
NEXT
*** Prints out the contents of the array
DEFINE nCount 1
DO WHILE nCount <=LEN(theclasses)
TEXT
<theclasses[nCount,1]> <theclasses[nCount,2]>
<theclasses[nCount,3]>
ENDTEXT
DEFINE nCount nCount + 1
ENDDO
```

The resulting array (stored in memory) is:

1 Executive	flat
2 Clerical	multiple
3 Maintenance	flat

While cycling through the database to determine if the code variable is “multiple” or ‘flat”, the data is written to an array. The next time that class number, description or code is required, the array can provide the data even though the "`SELECT * from class`" role has been closed (by using NEXT).

Note: You cannot use the DEFINE function to copy an array. For example, the following code would not be valid:

```
DEFINE Array2 Array1
```

See also: *TEXT With Array References* on page 85.

Changing Array Elements

You can use IStream Document Manager to update the value of an array element. Array elements are just another type of variable, therefore it is possible to update (redefine) the value of an array element using the DEFINE rule.

Example: `DEFINE theclasses[2,2] "Customer Service"`

This example changes the value of row 2 and column 2 from `Clerical` to `Customer Service`.

Chapter 4

Creating Model Documents

This chapter explains the main tasks you perform when using IStream Author. It is assumed that, as an author of model documents, IStream Author is your main tool, while the IStream Document Manager Repository is what you use to manage the storage of the your model documents.

IStream Author relies on the authoring language for generating model documents. This language consists of various rules and functions for governing how data is retrieved from an InfoSource and used in a document during assembly. For more information about creating InfoSources and using Assembler to generate documents, see *IStream InfoConnector* and *IStream Assembler Online Help*.

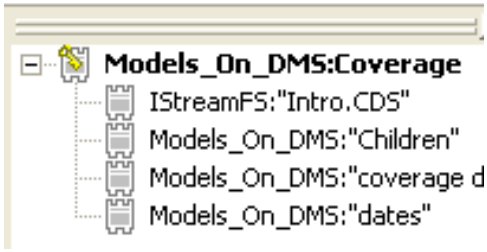
This section describes:

Model Documents

Your main task as an author is to develop model documents that are used for generating new documents, such as complex contracts, policies, proposals, and correspondence.

Model documents consist of different components, including master sections and sub-sections that determine the setup of a document, the data being used, and the format. When you generate a model document, IStream Assembler uses all of the sections to create the final, generated document.

When a model document is constructed with references, rules, and included sections, it is similar to a tree with many branches. The model document is like the main “trunk “of the tree. The sub-sections are like branches growing from the main trunk.



To generate a new document from a model document, Assembler goes through all of the required branches of the tree, locating the information needed by each rule and function. It then assembles this information into a complete, generated document that can be viewed in Word and eventually sent out electronically.

Related Topics

- *Building Model Documents and Sections* on page 156
- *Model Document Considerations* on page 165
- *The Authoring Language* on page 38

Model Document Components

A model document is a complex document, consisting of a master section and possibly one or more sub-sections created in IStream Author.

- **Master Section -** Master sections contain the key data definitions and certain other properties that allow for the assembly of documents. Each model document contains one master section and possibly other sections, called sub-sections. When you create a model document, it automatically creates the master section with the same name as the model document. You can edit master sections in the same way as sub-sections.
- **Content** - The content of a model document (the text, graphics, and borders) is in Word format. Content that is created in the model document remains the same but it may not appear in the generated document, depending on the defined rules and functions.
- **Rules and Functions -** Rules and functions are the coding elements that specify how the document is to be assembled from the section. Rules and functions produce different results depending on the rule and data values.
- **Sub-sections** - Sub-sections are all the sections (other than the master section) included in the model document. These sub-sections contain the formatting, the text, the graphics and some coding. Both the master section and sub-sections can use the INCLUDE rule to add a sub-section to a model document.

The first step in creating a model document is to create a master section. This master section includes references to sub-sections that can be used to make up the generated document, and these sub-sections can also reference other sub-sections, and so on.

The saved attributes in each section are:

- Word’s native representation.
- its compiled representation.
- a list of active InfoSources.
- its properties such as its name, description, and its effective and termination dates

Establishing a Repository

Repositories are the physical areas where model and generated documents are stored. Before creating a model document, you must first establish a repository for the document. This repository can be:

- an IStream Document Manager repository
- an IStream Document Manager FS (File System) repository

An InfoSource must be created to identify each repository available to use with IStream. IStream InfoConnector is used to define Infosources. Refer to the *InfoConnector Online Help* for more information.

You can set the default repository using IStream Author or IStream Assembler.

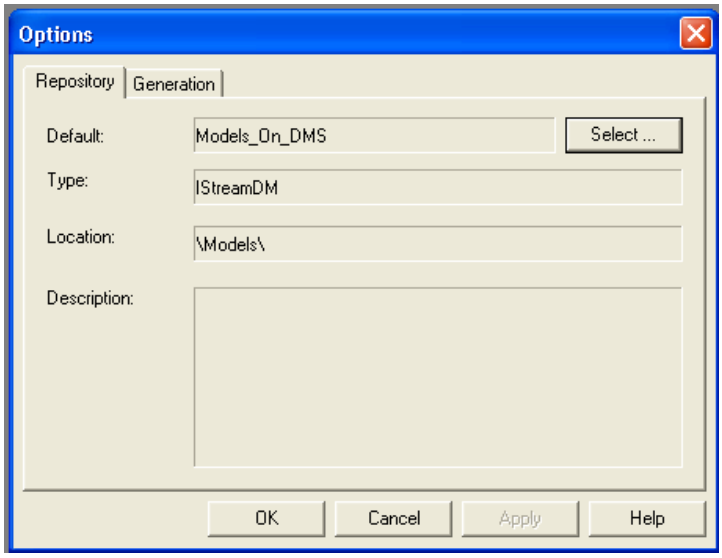
Setting the Default Repository

If you want to use the change the default repository, you can change it using IStream Author or IStream Assembler.

Method: Set the default repository using IStream Author

---

1. From the **Author** menu, select **Options**.
2. In the **Options** dialog box, click **Select...** to see a list of available repositories.



3. Select the repository where you want to store your documents.
4. Click **OK** to set this repository as the default and return to the main IStream Author window.

   When you open or save model documents and sections, this repository will now be selected. However, you can still choose a different repository if necessary.

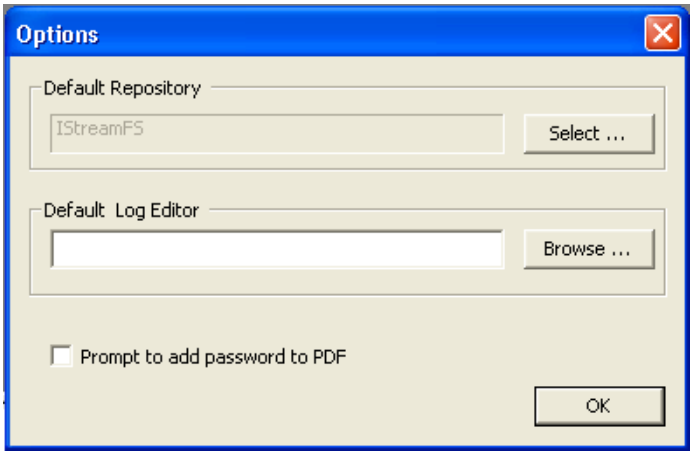
Note: You cannot use **Save As** for a model document on a DMS repository.

You can create new repositories: see *Adding an InfoSource* in the *InfoConnector Online Help.*

Setting the Default Repository Using Assembler

Method: Set the default repository using Assembler

1. Click **Start** > **Programs** > **IStream**> **Assembler**.
2. From the **Assembler** menu, select **Options**.



3. Click **Select** to see a list of the available repositories.
4. Click the repository where your documents are currently (or will be) stored.
5. Click **OK** to set this repository as the default and return to the main Assembler window.

   When you open or save model documents and sections, this repository will now be selected. However, you can still choose a different repository if necessary.

Note: You cannot use **Save As** for a model document on a DMS repository.

Building Model Documents and Sections

IStream Author allows you to build your model documents with or without the model document outline feature. Using a model document outline simplifies the authoring process, and gives you better control over the inclusion of IStream sections in your generated IStream documents. This section explains the tasks involved in building model documents and creating sections when you choose not to use the model document outline feature. For more information on model document outlines and how to use them, see *The Model Document Outline* on page 189.

This section describes:

- *Creating a New Model Document* on page 156
- *Creating New Sections* on page 161
- *Working with Existing Documents* on page 163
- *Model Document Considerations* on page 165
- *Viewing and Editing Section Properties* on page 165
- *Compiling Sections and Master Sections* on page 167
- *Printing a Section* on page 170

See also

- *The Authoring Language* on page 38
- *Model Documents* on page 152

Creating a New Model Document

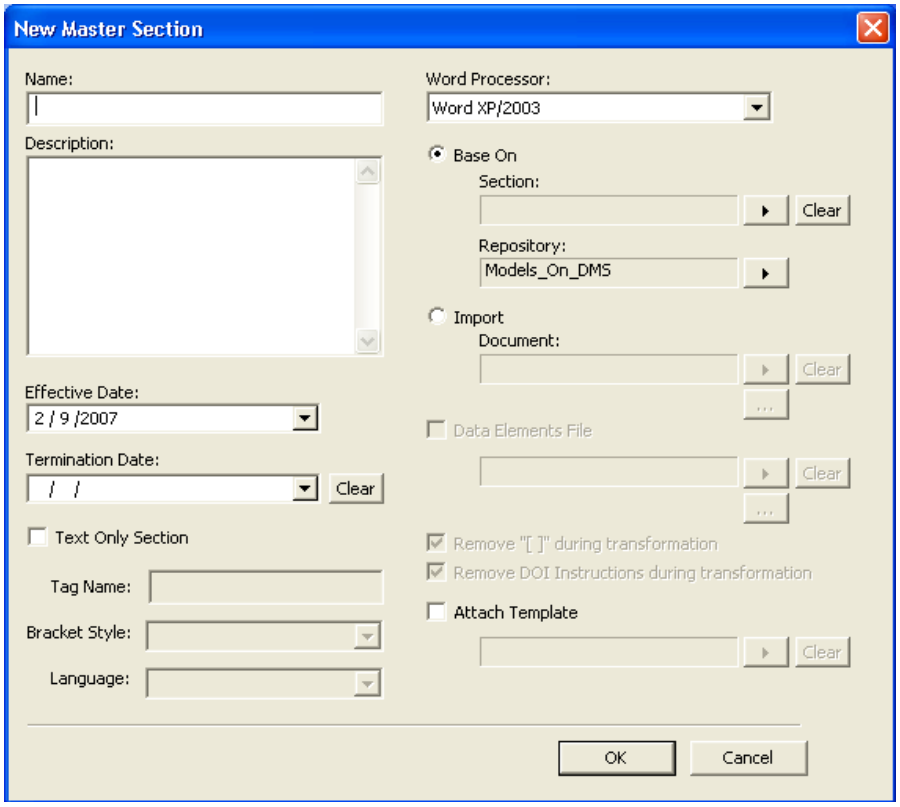
When creating model documents, you should create a Microsoft Word template and use styles to ensure that your documents are consistent. When designing your template, use as few styles as possible to keep the template simple.

Note: The procedure below describes how to create a traditional model document. For information about how to create an model document outline, see *Creating a New Model Document Outline* on page 192.

Method: Create a new model document using IStream Author

1. Ensure that Word is closed and then open IStream Author.
2. From the **File** menu, choose **New Model Document**.

   The **New Master Section Properties** dialog box opens. (A master section must be created for each new model document.)



3. Give the model document a **Name**. The master section name and model document name are the same. The name can be up to 254 characters long. However, names longer than 128 characters do not appear in the **Most Recently Used** list on the **File** menu.

Warning: Use only letters and numbers and do not use spaces or special characters (such as ? : / ' or -) when naming model documents and sections because this can cause problems when generating the model document.

4. Optionally enter a **Description** for the model document.
5. Enter an **Effective Date** if necessary. The default is the current system date.
6. Enter a **Termination Date** if necessary. The default is blank, meaning that this section is effective indefinitely. For more information on using these dates to control sections, see *Using Effective and Termination Dates* on page 160.
7. In the **Word Processor** field, select the version of Word you are using. (You probably have a default version of Word if you only have one version installed on your system.)

Important: All sections (master and sub-sections) in a model document must be in the same version of Word, otherwise problems can occur during generation.

8. Select **Based On** to base this master section on any other section. Click the browse arrow to locate a section on which you want to base your new section.

9. A **Repository** is selected. Click the browse arrow to change the repository. You must select from the repositories that have been set up. To set up a new repository, see *Adding an InfoSource* in the *InfoConnector Online Help*.

10. Click **Import** to import an existing Word .DOC file into your section. This option is different from the **Based On** selection because it allows you to use a native Word format file rather than an existing section.

11. Leave the **Text Only Section** check box clear. Never tag master sections as **Text Only**. If you select this check box, the section is evaluated as text only, causing the rules and functions to be ignored during generation. For more information on **Text Only** sections, see *Using the Text Only Option* on page 162.

Note: The following fields are active only if you are using IStream Writer:

- **Data Elements File**
- **Remove "[]" during transformation**
- **Remove DOI instructions during transformation**

  See the *Writer User Guide* for more information.

12. You may attach a template (.DOT file) to the master section. All sub-sections under this master section will use the template.The template must be in the local file system:

    a. Click the **Attach Template** check box.

    b. Click the browse arrow to navigate to the template to be attached.

13. Click **OK** to create the new master section.

    IStream Author displays the contents of your **Import**ed or **Based On** document if you used an existing document, or a blank document in the **Editor** window, ready for input and editing. Once you save or compile the new master section, the name of the new model document appears in the **Outline** window.

Note: If you are using key data, remember to define its properties in the master section properties.

Important: When you save your document, note that filenames for IStream documents may contain a maximum of 64 characters.

Defining Key Data

Key data identifies what initial data a model document needs to generate. It is defined in the section properties of a master section of a model document. A value for every piece of key data must be specified at generation time, and any of these values can be defined as persistent. Key data may identify the appropriate records or items in a particular InfoSource. Therefore, different key data can cause a section to assemble in different ways.

Examples of key data are:

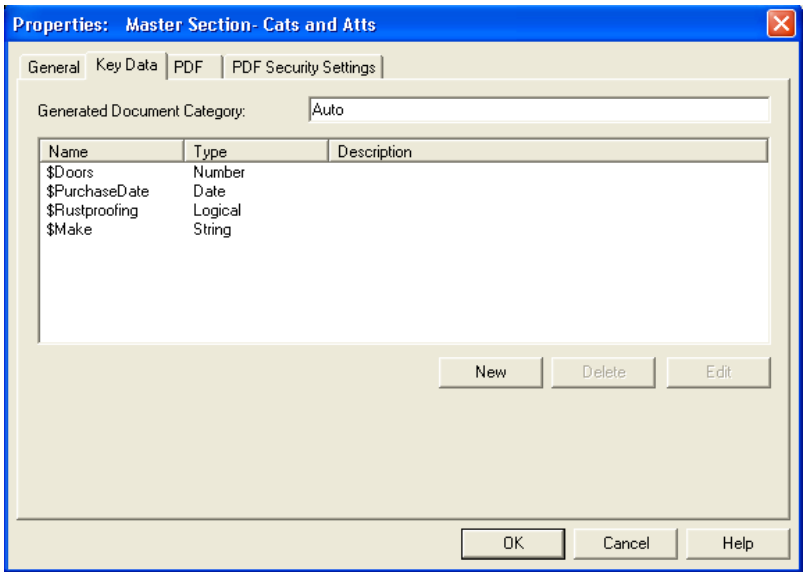
- **Insurance** - a policy number or policy holder
- **Banking** - an account number, statement date or account type
- **Correspondence** - a recipient's name and address

You are prompted for key data at every Assembler generation or regeneration. However, you are only prompted for persistent variables, which are a special type of key data, at the first generation. The persisted values are reused in all subsequent generations. For information on changing the persistent variables, see *Changing Persistent Variables* on page 31 of the *Assembler User Guide*.

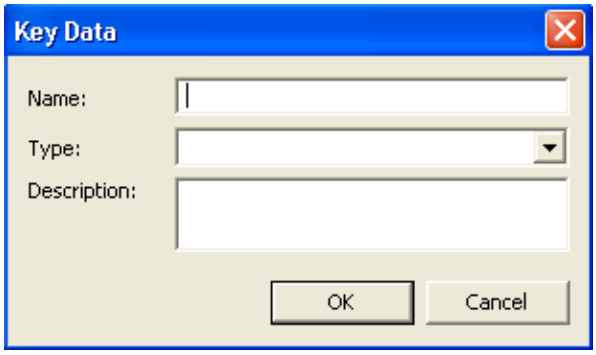
Note: You can also set up user prompts to open the key data prompt during Assembler. For more information, see *Setting Up User Prompts* on page 160.

Method: Define key data

1. Open the master section of your model document.
2. From the **File** menu, select **Section Properties**.
3. On the **Properties** dialog box, click the **Key Data** tab.



4. Click **New** to open the **Key Data** dialog box.



5. In the **Key Data** dialog box, define the properties for your key data, then click **OK**.

Important: Ensure that you do not use special characters in the key data **Name** or **Description**.

If a model document's key data is defined, when the master section is used to assemble a document, a **Key Data** window opens allowing the user to enter the value of the key data. Once the key data has been entered, it becomes read-only for that generated document.

Setting Up User Prompts

Sometimes, you may require information for your IStream documents that is not stored on a database. In these cases, you need to pause generation and enter the information manually so that the IStream document is complete, and includes the data specific to that generation. By using the PROMPT function while authoring model documents, an author can force Assembler to pause during generation and prompt the user to complete the required information.

If you know that a change is anticipated, but the exact details of the change are unknown, you can also ensure that the required information is entered at generation time. For detailed information about the PROMPT function, see *PROMPT* on page 131.

Using Effective and Termination Dates

Effective and termination dates control when certain sub-sections are used during assembly. By establishing the date ranges for information, you have greater flexibility in the options to include in your sections.

For example, you can set effective dates for a new regulation to take effect. When the effective date arrives, the information is automatically inserted in documents generated from the effective date onward.

When you create a new model document or section, you can set up your effective or termination dates as part of the model document or section properties. For more information, see *Creating a New Model Document* on page 156.

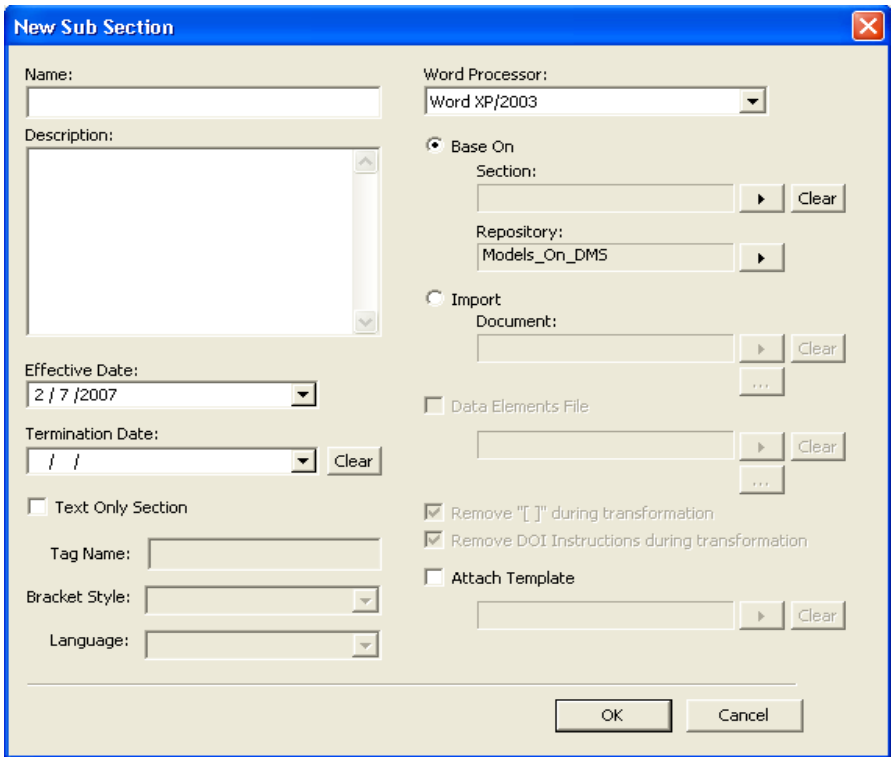
Once you set effective and termination dates you may need to change them later. For information on viewing and editing section properties, see *Viewing and Editing Section Properties* on page 165.

Creating New Sections

Method: Create a new section

1. From the **File** menu, choose **New Section**.

   The **New Sub Section** window opens.



2. Enter a **Name** for your section.

Warning: Do not use spaces and special characters (such as ? : / ' or -) when naming model documents and sections because this can cause problems when generating the model document.

3. Enter a **Description** for your section.
4. If required, enter an **Effective Date** or **Termination Date**.
5. Select the **Word Processor** you are using to create your section. Remembering that your sub-sections must use the same version of Word as the master section.
6. If you are basing your section on an existing section, enter the path for that section in the **Base On** field.
7. In the **Base On** section, accept the **Repository**, or click the browse arrow to select a new repository.

8. Click **Import** to import an existing Word file into your section. This option is different from the **Base On** selection because it allows you to use a native Word format file rather than an existing section.

Note: The following fields are active only if you are using IStream Writer:

- **Data Elements File**
- **Remove "[]" during transformation**
- **Remove DOI instructions during transformation**

  See the *Writer User Guide* for more information.

9. You may attach a template (.DOT file) to the section. The template must be in the local file system:

   a. Click the **Attach Template** check box.

   b. Click the browse arrow to navigate to the template to be attached.

10. If this section does not require any rules, click the **Text Only** check box. The content of the section will be generated as if TEXT/ENDTEXT surrounded it. Do not add TEXT/ENDTEXT to this section.

    When you select the **Text Only** check box, the following fields become available:

    - **Tag Name** - Specify a tag name if you want to use Customizer to develop custom wordings for this section later.
    - **Bracket Style** - Content enclosed within the bracket style that you choose is evaluated functions when the section is generated. If you do not choose a bracket style, angle brackets (< >) are used by default.
    - **Language** - Setting a language for your section helps to ensure that it is only included in IStream documents generated in the same language as the section. You must use the SETLANGUAGE function in another part of your document to take advantage of this feature. See *SETLANGUAGE* on page 139 for more information.

11. Click **OK** to create the new section.

Important: When saving a section of a document, note that filenames for IStream documents may contain a maximum of 64 characters.

- *Viewing and Editing Section Properties*

Using the Text Only Option

This check box indicates whether or not the section should be treated as text only. Only functions that are enclosed in brackets (< >, [ ], or { }, depending on the option that you choose) are compiled and evaluated during generation of a **Text Only** section. Establishing **Text Only** sections speeds up generation and makes it easier to create text sections, since TEXT...ENDTEXT rules are not required in a Text Only section.

The **Text Only** option is useful for debugging sections. You can change the properties for any section to **Text Only** instead of commenting them out or removing them from the model document. When you compile, all the rules and functions are visible in the compiled document so that you can narrow down which sections may have incorrect syntax or invalid rules.

Method: Select the Text Only option

1. From the **File** menu, choose **Section Properties**.
2. In the **Properties** dialog box, click the **Text Only** check box.
3. If you want to use Customizer later to create custom wordings for this section, specify a **Tag Name**.
4. Optionally, choose a **Bracket Style**. Content enclosed in the type of brackets that you choose will be processed as functions when the section is generated as a part of a model document. If you do not specify a bracket style, angle brackets (< >) will be used by default.
5. You can also choose a **Language**. When used in combination with the SETLANGUAGE function, choosing a language for your section ensures that it will only be included in documents generated in the language of the section. For more information on SETLANGUAGE, see *SETLANGUAGE* on page 139.
6. Click **OK**.

Working with Existing Documents

You can work with existing documents or their contents using any of the following methods:

- *Importing an Existing Document* on page 163
- *Inserting Existing Document Contents* on page 164
- *Copying, Cutting and Pasting Sections of Existing Documents* on page 164
- *Opening an Existing Model Document* on page 164
- *Opening an Existing Section* on page 165

Importing an Existing Document

1. From the **File** menu, choose **New Section** to create a new section.
2. Enter the properties of the new section.
3. Click **Import** to use a Word .DOC, .CMS, .CDS, or.CLG file in your section.

Note: To **Import** a .CMS, .CDS, or .CLG, the file must be in the new IStream Document Manager 6.1 format.

4. Click the browse arrow to locate the document, then select it. If you want to import a .CMS, .CDS, or .CLG, you need to change the file extension selected in the **Type** drop-down to *.* to see all available files.
5. Click **OK** to create the new section.

   The new section opens in the **Editor** window, ready for editing.

Inserting Existing Document Contents

1. From the **File** menu, choose **New Section** to create a new section.
2. From the **Insert** menu, choose **File**.
3. Locate the file containing the contents you are going to use.
4. Click **OK** to insert the entire contents of the document in your section.

Copying, Cutting and Pasting Sections of Existing Documents

1. Open an existing model document section.
2. Using your mouse, drag to select the portions of the document you want to copy or cut.
3. Once the area is selected, choose **Edit > Copy** or **Cut**.
4. From the **File** menu, choose **New Section** to create a new section.
5. From the **Edit** menu, choose **Paste** to insert the contents you copied or cut from the existing document.

Opening an Existing Model Document

- From the **File** menu, choose **Open Model Document**.

Note: When you open a .CMS file created in Author 5.x or earlier, the file will be converted to the new IStream Document Manager 6.1 format. If you close the document without saving, it will remain in the old format. Save the section to maintain the new 6.1 format.

Opening an Existing Section

Method: Open an existing section

1. From the **File** menu, choose **Open Section**.
2. Select from the list of **Recent Sections** in the **File** menu. If the section you want is represented in the tree view as part of the open model document you can double-click its name, or right-click the section name to open it.

Model Document Considerations

Please note the following information when working with model documents:

- model documents cannot contain nested Microsoft Word tables (tables within tables
- if a document has Track Changes on and uses protected sections, you need to accept all the changes before exporting it to PDF

Viewing and Editing Section Properties

Once you create a section, you can view or edit its properties. You can also view and edit the text, rules and functions you entered by compiling the document using the **Text Only** option. For information, see *Using the Text Only Option* on page 162.

Viewing the properties of a section allows you to change the key data (in a master section) or the effective and termination dates, and check the syntax of your sections to ensure that you can successfully compile and generate the document.

When working on multiple sections simultaneously, you can move between windows by changing the active section window.

Method: Display the section properties

- Click **File** > **Section Properties** or click the **Section Properties** button on the toolbar.

Viewing and Editing General Properties

From the **General** tab of section properties, you can:

1. Change the **Name** of the section.
2. Enter or change a **Description** of the section.
3. Change the **Effective Date** and **Termination Date**: see *Changing the Effective and Termination Dates* on page 166.
4. View the **InfoSources** that this section references.
5. Select the **Text Only Section** check box to indicate that the section is a text-only section: see *Using the Text Only Option* on page 162.

Viewing and Editing Key Data

From the **Key Data** tab (available for master Sections only) you can add, remove and edit the Key Data. You can also define **Generated Document Categories** that are used as metadata for searching within the DMS. See *Editing the Key Data* on page 167.

Changing the Effective and Termination Dates

If you need to change the effective and termination dates, you can do this in the **Section Properties** dialog box.

Method: Change the effective and termination dates:

---

1. From the **File** menu, choose **Section Properties** or click the **Section Properties** button on the toolbar.

2. Enter or select new dates from the **Effective Date** and **Termination Date** calendar boxes.
3. Click **OK** to save your changes.

Editing the Key Data

Once your model document or section is ready for compiling, you may realize that the key data you defined is incorrect. If so, you can change or delete it, before or after you compile.

Method: Edit the key data

1. Open the master section of your model document.
2. From the **File** menu, choose **Section Properties** or click the **Section Properties** button on the toolbar.

3. Click the **Key Data** tab.
4. Select the key data item you want to edit, then click **Edit**.
5. In the **Key Data** dialog box, change the key data, then click **OK**.

   To delete a set of key data, select the key data from the list, then click **Delete**.
6. Click **OK** to close the properties dialog box.

Changing the Active Section Window

If your model document has more than one section, you may need to switch between them to perform different functions. However, it is important to know which section is active before making changes, because only one section can be active at a time.

Method: Make a section active

- Click in the body of the section.
- Select a section's window from the **Window** menu. The name of the section appears in the title bar.

Compiling Sections and Master Sections

Compilation ensures that the rules you have in the model document are created correctly. You can compile using the **Author** menu or toolbar. A master section or IStream sub-section is also compiled automatically when you **Save**. After you have complete all your changes, and are ready to generate your final document, you must compile to ensure that your changes are correct.

Important: If you are using hidden text in Word, then under **Tools** > **Options** > **View**, ensure that the **Hidden text** check box is selected so that hidden text is displayed.

Otherwise, if a compilation error occurs in the hidden text, then you will not be directed to the correct text.

Method: Compile a section

Complete this procedure to compile a specific section.

- From the **Author** menu, choose **Compile Section** or click the **Compile** button on the toolbar.

Method: Compile all changed sections

Complete this procedure to compile all sections that have been edited and are currently open.

- From the **Author** menu, choose **Compile All Changed**.

Method: Rebuild your model document

Complete this procedure to recompile your master section and its sub-sections.

- From the **Author** menu, choose **Rebuild Model Document**.

Method: Check your compilation

To see if your compilation was successful, read the contents of the compile log in the **Compile/Generation Log** window at the bottom of the screen. You can double-click any errors to jump to the place where they occur in the model document.

Note: The generation log will not display the detailed output when the `SHOWRULES` function is enabled. To view this detailed output, you need to select **View Generation Log**.

The Compile Log

IStream Author creates a compile log each time a section or model document is compiled. The **Compile/Generation Log** window displays the compilation results and any error or information messages.

Compilation converts the Word-processed section into a format that IStream Assembler can read. During compilation, the system creates a tree view of the model document and its sections. All of the InfoSource references associated with any section are stored with that section.

Compiling performs a limited syntax check of the rules and functions, but it does not validate any expressions or connect to any data source at that time. The compile log will inform you of some of the most common syntax errors if they are found in your document, and tell you where in the document the error was found by providing a paragraph number. You can double-click on an error to jump to the place in the document where it occurred.

Compilation does not recognize InfoSources that have been referenced using the | (single pipe) or || (double pipe) syntax. These references appear in the tree view; however, double-clicking a section will not open it. These sections will still perform as expected during generation of IStream documents.

You can show or hide the log by choosing **View > Compile/Generation Log** from the **Author** menu or clicking the **Toggle Compile/Generation Log** button on the toolbar.

If required, you can also save and print compile logs.

Saving the Compile/Generation Log

Because the compile/generation log is overwritten every time you compile a section or model document, and every time you generate a new IStream document from Author, you may want to save a copy of it. This is useful in recording and troubleshooting a particular compilation or generation. You can also send a copy of the log to Customer Support if necessary.

Method: Save the compile/generation log

---

1. From the **File** menu, choose **Save Compile/Generation Log...**
2. Navigate to the correct storage location for the log, and enter a **File Name**.

Note: The default file extension for the compile/generation log is CMP. You can open a CMP file in any text editor such as Notepad, or you can open the file using Microsoft Word.

3. Click **Save**. The log is saved to the location you specified.

Printing the Compile/Generation Log

If you want a printed copy of the compile/generation log, complete the following steps:

1. Click **File > Print Compile/Generation Log...**
2. Author opens a **Compile.log** file and sends the log to your default printer.

3. Close the **Compile.log** file after your compile/generation log has printed.

Note: If you chose a word processor to print the log in **Author Options**, choosing **File > Print Compile/Generation Log...** will open the log in the word processor that you chose instead of sending it to the default printer. For more information, see *Setting the Word Processor to Print the Log* on page 30.

Printing a Section

Model document sections can be printed for general review and to check the rules and functions. The section prints in its raw form, as a Word document, exactly the way you see it on screen. It contains all the references, rules, and functions, but does not contain the variable values retrieved from the database.

Note: To print an entire model document, you must print all the sections individually.

Method: Print a section

1. Open the section.
2. Click anywhere in the section window to make the section active.
3. From the **File** menu, choose **Print**, or click the **Print** button on the toolbar.

If you select the model document in the **Outline** window, and click **Print**, the master section prints. You cannot print the entire model document in one operation, unless it has only one section.

To print a model document in its final assembled form, you must first generate the model document, then print it. The document will contain the variable values retrieved from the database during generation.

Hyperlinks and PDF Bookmarks

In PDF documents, bookmarks and hyperlinks are useful navigation aids.

Hyperlinks

You can use hyperlinks in IStream Author to link:

- from one location to another within the same document
- from one document to another
- to a website
- to an e-mail address

To create a hyperlink, you use the Word functionality in IStream Author. See your Word documentation for information on how to create a hyperlink.

Bookmarks

A bookmark is an item or location in a document that you can identify and name for future reference. Bookmarks in PDF documents are special hyperlinks that appear in a PDF's bookmark pane. By clicking a bookmark, a user can navigate easily to a specific page in the PDF document.

Hyperlink and Bookmark Considerations

Please note the following information when using hyperlinks and bookmarks:

- Microsoft Word bookmarks are different from PDF bookmarks and are not supported in IStream Author.
- The following options can be selected when rendering a PDF from a model document:
  - With Hyperlinks Preserved
  - With Hyperlinks Preserved and PDF Bookmarks Created
  - With PDF Bookmarks Created
  - Without Hyperlinks Preserved and PDF Bookmarks Created.
- When rendering a document into a PDF, the table of contents in the document is updated automatically. This "update" function is active when IStream Author is installed. If you use an existing macro to perform this task and want to use only the macro, this function can be disabled by your system administrator.
- To add hyperlinks or bookmarks to model documents that were migrated from earlier versions of Word to later versions that IStream Document Manager supports, you must delete the existing table of contents and add it back to the model documents. This ensures the hyperlinks and bookmarks will work correctly.

Defining PDF Properties for a Model Document

For each IStream Author document, you can specify to create the PDF using hyperlinks and PDF bookmarks within the model document. You can also define which Word styles will be used to create bookmarks in the PDF that will be generated from the model document. Note that choosing to enable exporting to PDFs using hyperlinks and bookmarks will increase the time it takes to generate the PDF.

These settings override those defined as system defaults (see *Editing PDF System Defaults* on page 31), which you must define before defining hyperlinks and bookmarks in a master section.

Important: When you define PDF properties for a model document, they only apply to the Word template that is attached to the model document (styles are retrieved from the template attached to the model). If you later change the template attached to the model document, you will have to redefine the PDF properties. For more information see *Using Word Templates and Styles* on page 176.

Also, the template attached to a master section in a model document overrides any templates attached to sub-sections of the model document. If sub-sections are using a different template than the master section, the template for the master section is what will be used when the model document is generated. Bookmarks will be created according to the styles defined in the template attached to the master section.

Defining PDF properties for a model document is done only in the master section.

Note: Using hyperlinks and bookmarks in documents may result in longer generation times when these documents are generated.

Important: Bookmarks will not be generated for paragraphs with text containing more than one style in a line. You should therefore ensure that the text in each style is on its own line.

Method: Define PDF properties for a model document

1. Open or create a master section.
2. On the **File** menu, select **Section Properties**, or click the **Section Properties** button on the **Author** toolbar.

   The **Properties: Master Section** dialog box is displayed.
3. Click the **PDF** tab.

   The **PDF** tab is displayed with any previously defined properties for this particular model document shown in the **Include Style** list. Unused styles are shown in the **Don't Include** list.

Note: If you are defining properties for the first time, or no properties have been defined for the model document, then the system default styles are displayed: see *Editing PDF System Defaults* on page 31.

4. Select an option from the **Render to PDF** drop-down list.

5. Click a style in the **Don't Include** list to highlight it, then click <-- to add it to the **Include Style** list.

Tip: To add multiple styles to the **Include Style** list, use the **Ctrl** or **Shift** key while clicking on one or more styles.

You cannot add more than 100 styles in the **Include Style** list.

6. If you selected a style that you no longer want, or want to remove a style, click a style in the **Include Style** list to highlight it and then click --> to add it to the **Don't Include** list.

7. Click **Restore Defaults** at any time to restore all system defaults in the **Include Style** list.

8. Click **OK** to save your hyperlinks and bookmarks definitions and return to IStream Author, or **Cancel** to discard your changes.

Note: You can make changes on the PDF tab, use any of the other tabs on the **Properties: Master Section** dialog box, or click **OK** any time to save your changes and return to IStream Author.

Adjusting PDF Margins

Your IStream administrator can configure the default margin sizes used in the PDF files that are rendered from your model documents. Please contact your IStream administrator for assistance.

Related Topics

- *Editing PDF System Defaults* on page 31
- *Defining PDF Security Settings* on page 174
- *Defining PDF Properties for a Model Document* on page 172
- *Defining PDF Bookmarks in a Model Document* on page 174

Defining PDF Security Settings

The security options that you specify are applied when you export a .CLG to .PDF format in Microsoft Word. PDF security settings can help you to control user access and manipulation of generated IStream PDFs.

Method: Define the PDF Security Settings

1. With a master section open in Author, click **File > Section Properties > PDF Security Settings**.
2. To apply PDF security options to the model document, select the **Enable Security** check box. Selecting the check box encrypts the generated PDF, and any metadata in the PDF will not be accessible by search engines. You can now modify the security options that will be applied:
   - **Allow Content Copying and Extraction** allows users to select and copy the contents of the generated PDF document, and allows utilities, like Acrobat Catalog, to access the contents.
   - **Enable Content Access for the Visual Impaired** allows visually impaired users to use screen readers to view the document.
   - **Change Allowed** allows you to specify how the user will be able to modify the document.
   - **Printing** allows you to control whether the document may be printed or not. Low resolution printing prints the document as a bitmap image with a resolution no higher than 150 dpi. High resolution allows you to print the document at any resolution.
3. Make your changes, and then click **OK** to close the dialog and apply them.

Related Topics

- *Editing PDF System Defaults* on page 31
- *Defining PDF Properties for a Model Document* on page 172
- *Defining PDF Bookmarks in a Model Document* on page 174

Defining PDF Bookmarks in a Model Document

This section describes how to define in a model document the text or objects to transform into PDF bookmarks when the model document is rendered to a PDF.

Before completing this procedure, you should define the PDF properties for your model document: see *Defining PDF Properties for a Model Document* on page 172. If you do not define these properties, then the default PDF properties are used: see *Editing PDF System Defaults* on page 31.

Note: Bookmarks and the table of contents in a PDF document can be different based on the styles defined for each. The table of contents uses the Heading 1, Heading 2, and Heading 3 styles, but you can change this.

If you use field codes to create a table of contents, use the TOC field code with the `\h` switch to enable hyperlinks in the PDF.

Note that only the headings in the PDF's table of contents will be hyperlinks, not the page numbers.

Important: Bookmarks will not be generated for paragraphs with text containing more than one style in a line. You should therefore ensure that the text in each style is on its own line.

Method: Define PDF bookmarks in a model document

1. Open any model section that you want to define text or objects as PDF bookmarks for.
2. Review which styles are included in the PDF definitions for the model document by clicking **Section Properties** on the **File** menu or the **Section Properties** button on the **Author** toolbar.
3. Highlight any text or object in the section and select a style.
4. Repeat the previous step for the text and objects you want to define.
5. Save the model document.

Important: If you later change the current Word template associated with this model document, or associate a new Word template with it, then the styles you have selected may no longer apply and you will have to repeat this procedure.

Related Topics

- *Editing PDF System Defaults* on page 31
- *Defining PDF Security Settings* on page 174
- *Defining PDF Properties for a Model Document* on page 172

Using Word Templates and Styles

A template is a special type of Word document. Every Word document is based on a template. A template determines the basic structure for a document and contains document settings such as AutoText entries, fonts, keyboard shortcut key assignments, macros, menus, page layout, special formatting and styles.

Template files have .dot file extensions and contain the following types of formatting definitions:

- styles
- tabs
- margins
- indents
- page setups
- page size
- macros

You should use styles and attach templates to each of your model document sections. This ensures that the same formatting is used within all sub-sections of a model document, and within all of the generated documents that are produced from the model document. Formatting inconsistencies among sections can cause problems when the final document is assembled.

To ensure that the final output is correct, each of these sections must be based on the same template.

The Normal.dot Template

Unless a Word document gets a template attached to it by a user, it is automatically based on the `normal.dot` Word template. This is applies to all model document sections.

When you create a model document, you need your model document sections to use styles and formatting standards that are not in the default Word `normal.dot` template because:

- users who will be generating your model documents can easily update and overwrite `normal.dot`
- the `normal.dot` template on other computers will not contain the styles you have added
- replacing a `normal.dot` template with your version will overwrite style changes that others have made for their formatting purposes
- if someone accidentally deletes `normal.dot`, Word creates a new version that will not have the style changes you have made
- everyone has their own `normal.dot` template, therefore styles will not be consistent from one system to another

Storing a Model Document Template

It is important to decide where the model document template will be stored before creating a model document.

If multiple authors are working on the same model, the template should be stored on the network so that everyone can access the same version of it.

It is important to store the most recent version of a shared template in a location that can be accessed (a network drive location) by all of the authors working on the same model document. This is because the same template has to be used in all of the sections of a model document.

Method: Change the default storage location for templates

1. From the Word **Tools** menu, select **Options**.
2. Click the **File Locations** tab.
3. Select the **Workgroup Templates** type, then click **Modify**.
4. Navigate to the new template storage location then click **OK**.

   The new storage location appears in the dialog box.

Attaching a Template to a Model Document Section

Once a template has been created, you need to attach the template to each model document section so that the formatting is applied and available within the section.

Important: Attach the same template to each section for consistency in formatting.

Method: Attach a template to a document

1. With the document open in Microsoft Word, on the **Tools** menu, select **Templates and Add-Ins**.
2. Click **Attach**, and then locate and select the template file.
3. Select the **Automatically update document styles** check box. This ensures all of the styles in the attached template will be applied to the document.
4. Click **OK** in the **Templates and Add-Ins** dialog box.

Using Styles

Styles define the character and paragraph attributes that you want to assign to text. Word comes with a default set of styles that are defined in the `normal.dot` template and are therefore available in all documents.

Creating and applying styles to common headings and text eliminates the need to remember all the font and paragraph characteristics that you want to apply. You will need to create your own styles and save them within your template to implement your document standards in your model documents.

Note: PDF bookmarks will not be generated for paragraphs with text containing more than one style in a line. Therefore, to create PDF bookmarks, ensure that the text in each style is on its own line.

Character formatting attributes include:

- font name, size, style and color
- bold, italics and underlining
- small caps and character spacing
- hidden text and text effects

Paragraph formatting attributes include:

- spacing before and after a paragraph
- indents
- line spacing and alignment
- center hyphenation
- page break controls
- outline levels
- bullets and numbering
- pagination

Warning: Do not manually change a style's formatting, because this will cause inconsistent formatting in the generated document. The style formats in the generated document will reflect the formatting that you manually changed, instead of the formats in the attached template.

Using Authoring Assistance

The Authoring Language, or model document language, consists of various rules and functions which control how data is retrieved from an InfoSource and used in a document during assembly. (For a complete description of the Authoring Language, see *Authoring Reference* on page 37.)

IStream Author provides Authoring Assistance, consisting of the Rule Wizard, the Reference Wizard, and the Function Wizard, to help you insert rules, references, and functions in your sections. These Wizards give reminders about the correct syntax to use when building rules and functions.

This section describes:

Using the Reference Wizard

You can use the Reference Wizard to insert references to data or information items from:

- active InfoSources and local variables, which are already referenced in the open model document section
- installed InfoSources, which are configured on the system, as indicated in the `local.idb` file

Method: Insert a reference to an InfoSource

Complete these steps to insert a reference to an active or installed InfoSource.

1. Place the cursor in your model document where you want to insert a reference.
2. Choose **Author > Tools > Authoring Assistance > Reference Wizard**.

   The **Reference Wizard** dialog box opens.
3. Click **Active** or **Installed**:
   - **Active** InfoSources are InfoSources that are referenced in the current document
   - **Installed** InfoSources are InfoSources that are configured on your system.
4. Select the **InfoSource** you want to reference:
   - Select **Local Variables** if you want to add a reference to a variable that is in the active document.

     -OR-

- Select an actual InfoSource name if you want to add a reference to a data item within the InfoSource.

5. If enabled, select **Insert reference in <> brackets** if you want to include angled brackets around the reference. For example, if you want to include a variable in a `TEXT/ENDTEXT` block, you would select this check box.
6. Click **Next**.
7. If you selected **Local Variables**, in the **Local Variables** dialog box, select the variable you want to insert.

   -OR-

   If you selected an actual InfoSource name. the navigation dialog box corresponding to the selected InfoSource type opens. Select the data item you want to reference, for example, a table and field name.
8. Click **Finish**.

   The **Reference Wizard** inserts a reference into the model document.

Using the Function Wizard

You can use the **Function Wizard** to insert functions in your model document.

1. Place the cursor in your model document where you want to insert a function.
2. Choose **Author > Tools > Authoring Assistance > Function Wizard**

   The **Function Wizard – Step 1** dialog box opens.
3. If enabled, select the **Prefix function with CALL** check box to insert `CALL` before the function name. This checkbox is automatically selected for the functions that require it.
4. If enabled, select the **Insert in <> brackets** check box to insert angled brackets around the function. Bracket are required if this function is contained within a `TEXT/ENDTEXT` block.
5. Select a function **Category** and **Name**.
6. If parameters are required:
   a. Click **Next**. The **Function Wizard - Step 2** dialog box appears.
   b. Enter the parameter's value in the field at the bottom of the dialog box, or click **Ref** or **Fx** to insert a nested reference or function as a parameter. Click **Undo** to discard any changes to this field.
7. Click **Finish**.

   The **Function Wizard** inserts a function into the model document.

Inserting Nested References

While using other Authoring Assistance wizards, you can use the **Reference Wizard** to insert a nested reference to an active or installed InfoSource.

1. In the **Step 2** dialog box of the **Authoring Assistance** wizard, select a parameter that you want to set a value for.
2. Click **Ref** to open the **Reference Wizard**.
3. Complete step 3 onwards in *Using the Reference Wizard* on page 179. Follow the steps to insert a reference to a data item within the InfoSource.

Inserting Nested Functions

While using other Authoring Assistance wizards, you can use **Function Wizard** to insert a nested function. See *Using the Function Wizard* on page 180.

Using the Rule Wizard

You can use the Rule Wizard to insert rules with the correct syntax in the model document.

Note: Some rules open a condition that must be closed. For example, `TEXT` requires an `ENDTEXT`. When using a rule that requires a close rule, enable the check box at the bottom of the dialog box that includes the close rule.

1. Position the cursor in the model document where you want to insert a rule.
2. Choose **Author > Tools > Authoring Assistance > Rule Wizard**.

   The **Rule Wizard - Step 1** dialog box opens.
3. Select the rule you want to insert.

   Any options specific to the specific rule you selected will appear in the bottom of the dialog box. Select any options that you require.
4. If more information is needed:
   a. Click **Next**.
   b. If the rule you selected relates to an InfoSource, click **Ref** to open the **Reference Wizard** from the **Rule Wizard - Step 2** dialog box. See *Using the Reference Wizard* on page 179 for help.
   c. If you selected **`CALL`**, the **Function Wizard** opens. See *Inserting Nested Functions* on page 181 for help.
   d. Enter any information needed for the rule you are building.
5. Click **Finish**.

   The **Rule Wizard** inserts a rule into the model document.

Authoring Model Documents for Remote Editing

Model documents can be edited through the **Edit** function in IStream Document Manager. This function allows users to perform Authoring functions on model documents using the web interface of IStream Document Manager.

This section describes remote editing considerations and the steps required to create model documents that can be used with the **Edit** function.

Considerations for Remote Editing

You do not need to radically change your model documents to use the IStream Document Manager **Edit** function. However, you should be aware of the following considerations:

- If your model document contains subsections, they must all be in the same folder as the master document.
- When creating a model document, do not include subsections from other model documents.
- Only one master section should appear in your model. Do not include master sections from other model documents.
- Use the ".\" syntax in INCLUDE rules to specify relative paths for the included sections: see *Syntax: INCLUDE ".\directoryname\section_file"* on page 69.
- Do not use the IStreamDM InfoSource in INCLUDE rules.
- Use Word's document protection features to protect portions of the document that you do not want to be changed.

Model Documents and IStream Document Manager

When you store a model document in the IStream Document Manager, you can use the **Edit** command to open and edit master sections and sub-sections in Microsoft Word. You can use **Edit** on all of your standard documents, as well as on your IStream sections. After you are finished making the changes, you will be prompted to save the section back to the DMS. The standard add-ins supplied in previous releases are no longer needed. If you had customized add-ins that you were using previously for **Model Edit**, you will need to copy the templates to your local computer and attach them to your model document sections.

If you use the **Edit** command to modify an IStream section created in an earlier version of Author, the section is converted to the new IStream Document Manager format, and a version is saved to the DMS. The section is opened in Word in its new format for editing. When you close the section, you will be prompted to save it back to the DMS as another version.

After you have created your model document, you need to set some options for the model document in IStream Document Manager.

If you have not been saving your model document in the DMS repository during its development, save the model document to the DMS repository now.

With the model document in IStream Document Manager, you need to complete the following procedure to prepare it for use with the **Edit** function:

- *Setting Model Document Permissions* on page 183

Setting Model Document Permissions

For other users to be able to copy and edit model documents from the DMS, you need to grant permissions for users of the document.

Method: Set model document permissions in IStream Document Manager

1. Launch IStream Document Manager and navigate to the location of the model document.
2. From the **Functions** menu, select **Permissions**. The **Permissions** page displays.
3. For the users and groups who will be using the model document, assign the following permissions:
   - for **Groups** and **Users** that you want to see the model document, assign the **See** and **See Contents** permission: this also assigns the **Copy** permission
   - for **Groups** and **Users** that you want to generate new IStream documents from the model from this location, assign the **Generate** permission
   - for **Groups** and **Users** that you want to author the model document (using IStream Author or the IStream Document Manager **Edit** function), set the **Reserve** permission: this also assigns the **See**, **See Contents**, **Modify**, and **Delete** permissions.
4. Click **Update** to set the selected permissions.
5. Click **Done** when you are finished setting permissions.

Technical Requirements for Remote Editing

To allow users to author model documents through IStream Document Manager, certain settings and options need to be set:

- InfoSources referenced by remotely authored model documents must be duplicated on all computers running IStream Assembler for generation. This includes all computers running IStream Assembler — computers running the server version of IStream Assembler, and IStream Publisher Workers.
- The IStreamDM InfoSource must point to the root of IStream Document Manager.
- Use InfoConnector on each Assembler computer to create new InfoSources that point to the root of the IStreamDM InfoSource.

Generating IStream Documents from Model Documents

You can do a complete assembly of the model document by generating it. A generation of your model document may include prompts for and saving various types of key data.

Generating model documents involves:

- *Generating a New IStream Document* on page 185
- *Regenerating an Existing IStream Document* on page 186

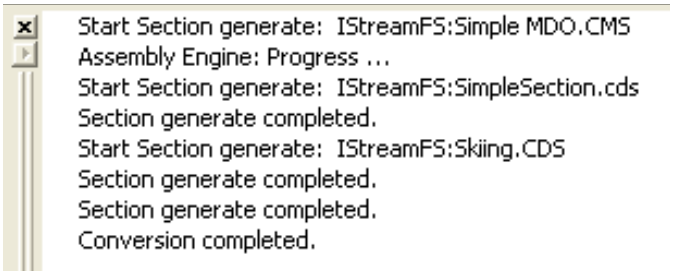
Generating a New IStream Document

Method: Generate a new document

1. Create the model document you want to generate.
2. With the model document open in the **Editor** window, select **Generate** > **New Document** from the **Author** menu or click the **Generate** button on the toolbar.

3. If you have not yet saved your model document, you are prompted to save it. Enter a name for your file that is the same name as your model document.

   If your variables are not set as persistent, you are prompted to define your key data, then generation begins.You can view progress messages in the log window in Author.



4. When the assembly is complete, Microsoft Word launches with the generated document open for you to view.

   After you have generated your document, you can double-click on any compilation errors to go to the place in the model document where they occurred. You may also want to save or print the generation log. Keeping a copy of the generation log can help you to identify and fix any problems that you may have encountered during generation. If you save a copy of the log, you can send it to Customer Support if necessary.

Note: The generation log will not display the detailed output when the SHOWRULES function is enabled. To view this detailed output, you need to select **View Generation Log**.

Method: Save the log

---

a. Click **File** > **Save Compile/Generation Log...**

b. Navigate to the correct storage location for the log, and enter a **File Name**.

Note: The default file extension for the compile/generation log is CMP. You can open a CMP file in any text editor such as Notepad, or you can open the file using Microsoft Word.

c. Click **Save**. The log is saved to the location you specified.

Method: Print the log

---

a. Click **File** > **Print Compile/Generation Log...**

b. Author opens a **Compile.log** file and sends the log to your default printer.

c. Close the **Compile.log** file after your compile/generation log has printed.

Note: If you chose a word processor to print the log in **Author Options**, choosing **File** > **Print Compile/Generation Log...** will open the log in the word processor that you chose instead of sending it to the default printer. For more information, see *Setting the Word Processor to Print the Log* on page 30.

5. From the **File** menu, choose **Exit** to close Assembler and return to IStream Author.

Regenerating an Existing IStream Document

Once you have created an IStream document, you can easily regenerate it while working with it in other applications:

- In Microsoft Word or Assembler, you can open and edit a .CLG, and regenerate the IStream document. See *Regenerating an IStream Document* of the *IStream Assembler Add-in for Microsoft Word* Guide, and *Regenerating a Document* in the *IStream Assembler Online Help* for more information.
- If your IStream document is stored in the DMS, you can regenerate it from the IStream Document Manager user interface. For more information, see *Regenerating an IStream Document* in the *IStream Document Manager DMS Guide for IStream and Model Docs*.

Note: If your IStream document was edited in IStream Customizer and contains custom wordings, when you **Export CLG to PDF**, the PDF created will not be regeneratable.

Chapter 5

The Model Document Outline

This chapter gives an overview of the model document outline and explains how to use this feature to create your model documents.

This chapter describes:

The Model Document Outline

The model document outline is a tool that you can use to create model documents quickly and easily in Author. Model document outline enables you to incorporate sections in your model document, and to specify conditions for their inclusion in generated documents without embedding INCLUDE rules in your IStream sections.

The model document outline is the master section of your model document in an easy to use outline format. Like any other master section, a model document outline has a `.CMS` file extension. Unlike with regular master sections, you use unique model document outline features to include and control IStream sub-sections. These unique features help to speed up the process of authoring, and limit coding errors.

From the model document outline, you create **section definitions** for the sub-sections in your model document. Each section definition that you create applies to one or more IStream document sections, and provides the conditions under which the IStream sections will be included in generated documents.

You can include multiple versions of the same section under one section definition, defining different **Effective** and **Termination Dates** for each. Only the version falling within the current date range will be evaluated by the section definition conditions during generation, while sections with dates outside of the appropriate range will be ignored.

IF groups give you even more control over when sections are included in your generated documents. You begin by grouping the chosen section definitions together. When the IF condition of the first section definition in the group is not met, Assembler will evaluate the IF rules applied to subsequent section definitions to decide which should be included in its place.

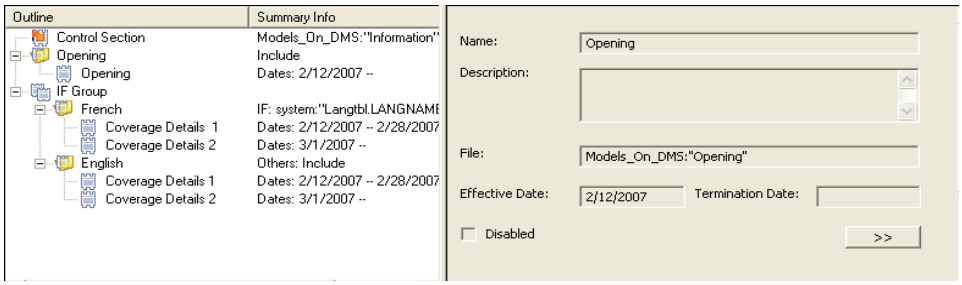
The Model Document Outline Interface

When the model document outline is open in the **Editor** window, you can see the model document's sections and section definitions in the **Outline** column. The **Summary Info** column gives some basic information about each section and section definition.

Icons next to each section and section definition help you to identify which items in the **Outline** are sections, and which are section definitions:

-  identifies the **control section**. The **control section** is at the top of the outline, with all sections and section definitions falling below it.
-  identifies an **IF group**. Some, but not all section definitions may belong to an IF group, depending on the design and purpose of your model document. You can use the control next to an IF group to expand and collapse the list of section definitions belonging to it.
-  identifies a **section definition**. Use the control next to each section definition to expand and collapse the list of sections belonging to it.
-  identifies a section. Each section falls below a section definition in the **Outline**.

The right-hand side of the model document outline screen shows you the current **Properties** of the section or section definition selected in the **Outline**.



You will also notice that an **Outline** menu is available when you have a model document outline open. The **Outline** menu commands enable you to add, edit and control sections and section definitions.

Working with Model Document Outlines

This section describes:

Creating a New Model Document Outline

With Author open on your computer, you can easily create a new model document outline. After creating the outline, you can edit its properties and save it.

Note: For information about how to create a traditional model document, see *Creating a New Model Document* on page 156.

1. From within Author, choose **File > New Model Document Outline...** , or click  on the toolbar.
2. In the **New Master Section** dialog, enter a **Name** for the model document outline. It can be up to 254 characters long. However, names longer than 128 characters do not appear in the **Most Recently Used** list on the **File** menu.

Warning: Do not use spaces and special characters (such as ? : / ' or -) when naming model document outlines and sections because this can cause problems during generation.

3. Optionally, enter a **Description**.
4. If necessary, modify the **Effective Date** for the model document outline. By default, the **Effective Date** is the current system date
5. If applicable, specify a **Termination Date**. By default, the **Termination Date** field is empty, meaning that the outline applies indefinitely.
6. You may attach a template (.DOT file) to the outline. All sub-sections under this outline will use the template. The template must be in the local file system:
   a. Click the **Attach Template** check box.
   b. Click the browse arrow to navigate to the template to be attached.
   c. Select the template and click **Open**.
7. Click **OK**.

Author creates a model document outline based on the options you specified.

Defining a Control Section

The control section is the pre-processing section of the outline. This section can be used to perform queries or define variables that will be used in processing the model document. A control section is optional. If you use a control section, it will always be processed first.

The control section icon always remains in the model document outline as a placeholder, whether you define a control section or not. If you want to use a control section, you attach the IStream section of your choice, or create a new section to act as the control section. If you do not attach a file to the control section, it will be skipped during document generation.

The following are important points to note about control sections:

- A control section cannot be moved in the outline – it must always be first.
- You cannot rename the control section.
- Since each outline can only have one control section specified, you cannot copy and paste a control section.

Method: Define a control section

1. In the **Editor** window, click the **Control Section** to select it.
2. Do one of the following:
   - Click **Outline > Properties**.
   - Right-click the **Control Section** and choose **Properties** from the context-sensitive menu.

   The **Control Section Properties** dialog opens.You can choose to define an existing IStream section as the control section, or you can create a new IStream section.
3. **If you want to define an existing IStream section as the control section:**
   a. Click  .
   b. Navigate to the section that you want to use. If necessary, click the **Repository...** button to select a different repository.
   c. Select the section and click **Open**.

Note: If effective and termination dates are specified for the section that you are defining as the control section, these dates will be ignored. The control section will always be included when generating IStream documents from the model document outline.

4. **If you want to create a new section to define as the control section:**

a. Click  .

b. In the **New Sub Section** dialog, specify a section **Name**, and modify any other settings as required. For more information on creating sub sections, see *Creating New Sections* on page 161.

c. In the **New Sub Section** dialog, click **OK**.

d. You are prompted to save the new section. Navigate to the specific location for the section, enter a **Name**, then click **Save**.

5. To save your changes to the control section, click **OK**.

The name of the file for the control section appears in the **Summary Info** column.

Related Topic

- *Removing the Attached Control Section File* on page 194

Removing the Attached Control Section File

Follow these steps if you have chosen a file to act as the control section and you want to remove it.

These instructions will remove the attached control section **File**, as well as any **Description** information that you may have included. If you want to modify either of these fields without deleting all of the information, select the control section and choose **Outline > Properties**.

Method: Remove the attached control section file and description

---

1. Select the control section in the outline.
2. Choose **Outline > Remove**, or right-click the control section and choose **Remove**.

Opening an Existing Model Document Outline

You can open an existing model document outline in Author for editing.

Note: If you open your model document outline in Microsoft Word, you will see text similar to this:

```
*** Outline: My MDO
*********************************************
***                                       ***
*** WARNING: Automatically generated code ***
***      Do not modify!                   ***
***                                       ***
*********************************************
```

You are able to view the model document outline's code, but you cannot make or save any changes. You must open the outline in Author and modify any rules using model document outline functionality.

Method: Open a model document outline

1. In Author, choose **File > Open Model Document**. You can also choose  from the toolbar.
2. If you already have a model document open in Author, you are prompted to close it. Click **Yes** to close and save any changes to the current model document. If you choose **No**, the current model document stays open and you cannot open another one.
3. Navigate to a `.CMS` file saved in the model document outline format. Click **Open**.

   The model document outline opens in the **Editor** window.

Note: If you selected a `.CMS` file that was not created in the model document outline format, it opens as a standard `.CMS` file in the **Editor** window.

Saving a Model Document Outline

1. Click **File > Save**, or click  on the toolbar.
2. If this is the first time that you are saving the outline, the **Save As** dialog opens. Do the following:
   a. Navigate to the location where you would like to save your outline. If necessary, click the **Repository...** button to select a different repository.
   b. Enter a **Name** for the model document outline and click **Save**.

      The model document outline is saved with a `.CMS` extension in the location that you specified.

Editing the .CMS Properties of a Model Document Outline

You can edit a model document outline's properties in the same way that you edit the properties of a standard IStream master section.

1. With a model document outline open in Author, choose **File > Section Properties**, or click  .
2. The **Properties** dialog opens. You can see and change information under the following tabs:
   - In the **General** tab, you can edit the model document outline's **Name**, **Description**, **Effective Date** and **Termination Date**. For more information on effective and termination dates, see *Using Effective and Termination Dates* on page 160.
   - In the **Key Data** tab, you can modify the **Generated Document Category**, create and delete key data, or edit existing key data. For more information, see *Defining Key Data* on page 158.
   - In the **PDF** tab, you can change the selected setting in the **Render to PDF** drop-down. For more information and instructions, see *Defining PDF Properties for a Model Document* on page 172.
   - In the **PDF Security Settings** tab, you can select or clear the following check boxes:
     - **Enable Security**
     - **Allow Content Copying and Extraction**
     - **Enable Content Access for the Visual Impaired**

   You can also change the option selected in the **Change Allowed** and **Printing** drop-down lists. For more information on these settings, see *Defining PDF Security Settings* on page 174.

Copying and Pasting Model Document Outline Elements

You can cut, copy and paste model document outline sections, section definitions and IF groups.

Method: Copy or cut and paste a model document outline element

---

1. Click the section, section definition, or group that you would like to cut or copy.
2. Complete any of the following steps:
   - Click **Outline > Copy or Outline > Cut**.

- Right-click an item and choose **Copy** or **Cut** from the context-sensitive menu.
- Select an item then click **Ctrl-C** to copy it or **Ctrl-X** to cut it.

3. Select a valid position in the **Outline** to paste the copied element.
   - If you copied a **section**, you must select a **section definition** in the outline. The section will become part of the selected section definition when you paste.
   - If you copied a **section definition**, you can select:
     - the control section. When you paste, the section definition will be placed just below the control section.
     - an IF group. When you paste, you can choose to place the section definition **Inside** or **Below** the IF group. Choosing **Inside** places the section definition at the bottom of the IF group.
     - another section definition. The definition that you copied will be added just below the selected section definition when you paste.
   - If you copied an **IF group**, you can select the control section, an ungrouped section definition, or another IF group. The copied IF group will be pasted just below the selected element.
4. Click **Outline > Paste**, or click **Ctrl-V**.
5. If applicable, choose whether you want to paste your section definition **Inside** or **Below** the IF group.

   The element is pasted.

Tip: To copy an element, you can also drag and drop it while holding down the **Ctrl** key. Using **Ctrl + drag and drop** creates a copy of the element and moves it to a new position on the outline instead of moving the original.

Moving Model Document Outline Elements

You can move a model document outline section, section definition, or IF group by dragging and dropping it into a new place on the outline. Click and drag the element that you want to move over another element in the outline. When you release the mouse button, the element you are moving is placed just below it.

The rules for where you can place certain elements in the outline are the same as they are for copying and pasting. For more information, see step 3 of *Copying and Pasting Model Document Outline Elements* on page 196.

Compiling and Generating from Model Document Outlines

You compile, rebuild, and generate from model document outlines the same way that you compile and generate from standard model documents. For more information, see *Compiling Sections and Master Sections* on page 167, *Generating a New IStream Document* on page 185 and *Regenerating an Existing IStream Document* on page 186.

Using Section Definitions

When you create your model document using the model document outline format, you use section definitions to create rules that determine when your IStream sections are included in generated documents. You can add multiple versions of the same section to a single section definition, and give each section a different effective date range. During generation, Assembler evaluates the applicable section according to the rule specified in the section definition. How Assembler evaluates the rule determines whether a given section will be included in the final, generated document.

Each time you add a new section definition, you must also add an IStream section that the rule in the definition will apply to. You can choose an existing IStream section to add to the definition, or you can create a new IStream section.

Adding a Section Definition

1. Do one of the following:
   - **Click Outline > Add Section Definition...**
   - Right-click on a section definition, group, or control section and select **Add Section Definition...**

Important: Make sure that you do not have a section selected in the **Outline**, or the **Add Section Definition...** option will be grayed out.

The **Add Section Definition** dialog opens.

2. Enter a **Name** for the section definition. If you leave the **Name** field empty, Author will import the name of the section that you attach to the definition later.
3. Optionally, enter a **Description**.
4. In the **Include Syntax** drop-down, choose one of the available options:
   - Choosing **Include** means that a section with the appropriate effective/termination date range will be included in generated IStream documents unconditionally.
   - Choosing **Query** enables you to set conditions for the inclusion of this definition's IStream section, based on information retrieved from a database. When using **Query**, the section will be included once for each record found in the database or XML file.
   - Choosing **Rule** enables you to set conditions for the inclusion of this definition's IStream section when database queries are not required to complete the rule.

   If you choose the **Query** or **Rule** option, you can use the Rule Builder or EasyText to compose your rule. You can also enter rules and queries manually.

Note: XML-based queries require two separate QUERY statements. If you are entering an XML query manually, you will need to use both available **Query** fields – one

for each statement. Author will insert the necessary BREAK and NEXT rules for you. For more information, see *Using QUERY with IStreamXML InfoSources* on page 78. If your query references a type of InfoSource other than XML, only complete the first **Query** field.

5. If you want to use the Rule Builder, click **Rule Builder**.

   - **For query-based rules:**

     a. Enter the database name or IStream XML InfoSource. You can click **Ref...** to get help with this from the **Reference Wizard**. For more information on using the **Reference Wizard**, see *Using the Reference Wizard* on page 179, steps 3-6.

     b. Click **Next**.

     c. Complete the **SELECT**, **FROM**, and **WHERE** fields. Again, you can use the **Reference Wizard** if you want assistance.

     d. Click **Finish**. Your query-based rule appears in the **Add Section Definition** dialog in the **Query** field.

   - **For rules that do not require a query:**

     a. Enter the conditional IF statement. You can use the **Reference and Function Wizards** to help you. For more information, see *Using the Reference Wizard* on page 179, steps 3-6, and *Using the Function Wizard* on page 180, steps 3-5.

     b. Click Finish. Your rule appears in the **Add Section Definition** dialog in the **IF** field.

6. If you want to use EasyText, click  . Choose an entry and drag it into the **Query** or **IF** field in the **Add Section Definition** dialog.

7. In the **Select Section** portion of the dialog, choose a section to add to the definition, or create a new section.

Note: If you click **OK** to create a section definition without first adding a section, Author applies a blank section to the definition. In the **Outline**, a section appears below the new section definition with a warning icon next to it. You will need to edit the **Properties** of this blank section later by choosing an existing IStream section or by creating a new IStream section to hold its place.

To select an existing section:

a. Click  .

b. Navigate to the section that you want to add to the definition. If necessary, click the **Repository...** button to select a different repository.

c. Select the section and click **Open**.

If you left the **Name** for the section definition blank earlier, the **Name** of the section that you selected will be used as the section definition **Name**. If you typed in a **Name** for the section definition, the information that you entered will not be overwritten when you choose a section.

To create a new section:

a. Click  .

b. In the **New Sub Section** dialog, specify a section **Name**, and modify any other settings as required. For more information on creating sub sections, see *Creating New Sections* on page 161.

c. In the **New Sub Section** dialog, click **OK**.

d. You are prompted to save the new section. Navigate to the specific location for the section, enter a **Name**, then click **Save**.

8. If necessary, modify the **Effective Date** and **Termination Date** for the section. The default **Effective Date** is the current system date. The **Termination Date** field is empty by default, meaning that the section will be effective indefinitely.

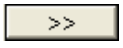
9. The **Disabled** check box allows you to ignore a section during generation without removing it from the model document outline. If you want the section to be ignored during generation, select the **Disabled** check box.

If you select the **Disabled** check box, the section will be grayed out in the outline.

10. Click **OK** in the **Add Section Definition** dialog.

The new section definition and section are added at the end of the **Outline**. By default, the name given to the section is the same name that you gave to your section definition. You can change these names individually later.

The **Summary Info** column gives you some basic information about the new section and section definition. For the section definition, the INCLUDE rule conditions are displayed. For the section, the effective and termination dates are shown.

The pane on the right displays the section definition or section **Properties** when you have one selected in the **Outline**. When you have a section selected, you can view additional properties by clicking 

.

Note: The **File**, **Effective Date**, **Termination Date** and **Disabled** fields can all be changed without opening the section for editing. See *Editing the Section Properties* on page 205 for instructions. If you want to modify the *additional* properties (**Text Only**, **Brackets**, **Tag Name**, and **Language Used**), you must open the section in Author for editing and then select **File > Section Properties**. See *Creating New Sections* on page 161 for more information about each of the additional properties.

Adding a Section to an Existing Section Definition

You can add a new or existing IStream section to a section definition that you have already created.

Method: Add another section to an existing section definition

1. Select a section definition in the **Outline**.
2. Choose **Outline > Add Section...**, or right-click the section definition and choose **Add Section...**
3. Enter a **Name** for the section. If you leave the **Name** empty, Author will later import the **Name** of the section that you attach.
4. Optionally, enter a **Description**.
5. You can choose to add an existing section to the definition or to add a new section.

   If you want to add an existing section to the section definition:

   a. Click  .

   b. Navigate to the section that you want to add to the definition. If necessary, click the **Repository...** button to select a different repository.

   c. Select the section and click **Open**.

   If you want to create a new section to add to the definition:

   a. Click  .

   b. In the **New Sub Section** dialog, specify a section **Name**, and modify any other settings as required. For more information on creating sub sections, see *Creating New Sections* on page 161.

   c. In the **New Sub Section** dialog, click **OK**.

   d. You are prompted to save the new section. Navigate to the specific location for the section, enter a **Name**, then click **Save**.
6. If necessary, modify the **Effective Date** and **Termination Date** for the section. The default **Effective Date** is the current system date. The **Termination Date** field is empty by default, meaning that the section will be effective indefinitely.
7. The **Disabled** check box allows you to ignore a section during generation without removing it from the model document outline. If you want the section to be ignored during generation, select the **Disabled** check box.
8. Click OK in the **Add Section** dialog.

   The section is added to the section definition that you selected, appearing below it in the **Outline**.

The section definition is now a multi-section section definition, meaning that the section definition contains multiple section versions with different effective and termination dates for each.

Note: If a  appears next to your section, that means that there is a conflict in the effective and termination dates between versions. More than one of the versions applies over the same period of time. To fix this problem, right-click the section and choose **Properties**. Modify the **Effective Date** and **Termination Date** fields so that the effective period for the new section does not conflict with the effective period of other sections.

Modifying the Properties of a Section Definition

1. In the Outline, select the section definition whose Properties you want to edit.
2. Click **Outline > Properties**, or right-click the section definition and choose **Properties**.

   You can edit the **Name**, **Description** and rule for the section.
3. When you are finished editing the section definition properties, click **OK** to close the dialog and to save your changes.

Moving a Section Definition

- To move a section definition, drag and drop it from one location to another in the outline.

  Dropping on a control section or another section definition will place the section definition below the target. If you drop the definition on an IF group, you need to choose whether you want to place the section definition **Inside** or **Below** the group. You cannot drop a section definition on a section.

Note: If you move a section definition into the first or middle position in an IF group, you will need to add a rule to the section definition if one is not already applied.

Removing a Section Definition from the Outline

You can remove a section definition from the model document outline. Removing the section definition will also delete any sections belonging to that definition. For instructions on how to remove a single section only, see *Deleting a Section from the Model Document Outline* on page 207.

Method: Remove a section definition

---

1. In the **Outline**, select the section definition that you want to remove.
2. Click **Outline > Remove**, or right-click the section definition and choose **Remove** from the context-sensitive menu.

3. A message asks you if you would like to remove the section definition. To confirm the deletion and remove the section from the list, click **Yes**.

   Author checks and notifies you of any possible errors. The definition and the sections belonging to it are removed from the **Outline** view.

Note: If the section definition that you deleted was part of a group and contained the IF rule for that group, the next section definition in the group becomes the IF rule. If the section definition that you deleted was part of a group and was the ELSE rule for the group, the ELSE rule is not transferred to another section definition.

Working with Sections in the Model Document Outline

From the model document outline, you can open and edit your IStream sections. You can also edit the Properties of those sections, and move and delete sections from the model document outline.

Opening a Section for Editing

1. Select the section that you want to open in the **Outline** column.

Note: You may need to click  next to one or more of your section definitions to see the section that you want.

2. Do one of the following:

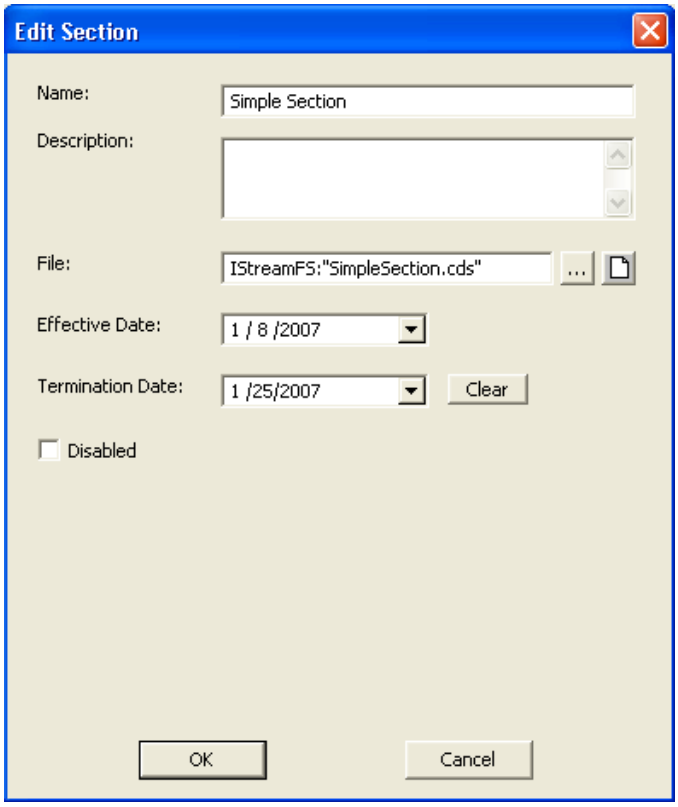
   - Right-click the section and choose **Open Section**.
   - Click **Outline > Open Section**.
   - Double-click the section in the **Outline** column.

   The section opens in Author for editing. To return to the model document outline without closing the section, double-click the outline in the **tree** view.

   You can also click  if you want to toggle easily between your IStream section and the model document outline.

Editing the Section Properties

1. In the **Outline** column, select the section whose Properties you want to modify.
2. Click **Outline > Properties**, or choose **Properties** from the right-click menu.



3. You can edit any of the fields in the **Edit Section** dialog. Make your changes, then click **OK** to save them.

Note: If you want to modify the **Text Only**, **Brackets**, **Tag Name**, and **Language Used** properties shown in the outline when you click >>, you must open the section in Author for editing and then select **File > Section Properties**. See *Creating New Sections* on page 161 for more information about each of the additional properties.

Moving a Section

You can move a section from one definition to another, or you can change the order of sections within a section definition.

Method: Move a section within a section definition

Edit the effective and termination dates for the section that you want to move. Author will reorder the sections in the section definition accordingly, placing the section with the earliest effective date range first in the list. For information on how to edit the dates, see *Editing the Section Properties* on page 205.

Method: Move a section from one section definition to another

Click and drag the section to the section definition that you want to move it to, and release the mouse button.

The section is moved to the section definition that you chose. Author evaluates the effective and termination date information to determine where the section should be placed in the definition's list of sections. You may need to edit the effective and termination dates of the section if there is a conflict in the effective date range between versions. For more information, see *Editing the Section Properties* on page 205.

Deleting a Section from the Model Document Outline

1. In the **Outline** column, select the section that you want to delete.
2. Do one of the following steps:
   - Choose **Outline** > **Remove**.
   - Right-click the section and choose **Remove**.
3. You are asked if you are sure that you want to remove the selected item. Click **Yes** to continue.

   The section is deleted. If the section was the only one belonging to a section definition, the section definition is also deleted. If more than one section belonged to the section definition, then Author deletes only the section that you chose.

Working with IF Groups

Model document outline uses IF groups to help you create complex conditions for the inclusion of your IStream sections without manually coding rules. When you add a section definition to a group, the rule applying to the first section that you add becomes the IF condition. The rules applied to the subsequent section definitions in the group become the ELSEIF conditions. If the last section definition in the group does not have a rule, an ELSE rule is applied to it.

Assembler evaluates the grouped sections in order. When the IF condition applied to the first definition is not met, Assembler evaluates the ELSEIF and ELSE rules on the subsequent grouped section definitions to determine which one should be included in the generated document.

Creating a New IF Group

After you have added section definitions to your model document outline, you can group section definitions together.

Method: Create a new IF group

---

1. Select the section definitions that you want to group by holding down the **Ctrl** key and clicking on them.

Tip: To select many section definitions at once, hold the **Shift** key and click the first and last section definitions that you want to group.

Important: You must select contiguous section definitions to group. You cannot skip any section definitions. If you need to move section definitions in the **Outline**, you can click on a section definition's name and drag it to a different place in the **Outline**.

2. Choose **Outline > Create IF/ELSEIF Group**, or right-click any of the selected section definitions and choose **Create IF/ELSEIF Group**.

   The section definitions that you selected are grouped. If you want to rename the **IF Group** to something else, you can right-click it and choose **Rename** from the context-sensitive menu.

   The rule applying to the first section definition becomes the group IF rule. The rules applied to other definitions in the group become ELSEIF rules. If the last definition in the group has no rule conditions, an ELSE rule is applied to it.

Note: All definitions except for the last one in the group must have rules applied to them. If a grouped section definition does not have a rule, a warning icon will appear next to it. To fix the error, right-click the section definition, choose **Properties**, and create a rule for the section definition.

You cannot place a section definition with a **Query** type rule applied inside of an **IF Group**. You will always receive an error.

Ungrouping Section Definitions

These instructions explain how to ungroup all section definitions and remove the IF group. To move section definitions without removing the group, see *Moving a Section Definition* on page 203.

Method: Ungroup section definitions and remove the group

1. Select the group that you want to remove in the **Outline**.
2. Choose **Outline > Ungroup**, or right-click the group and choose **Ungroup** from the context-sensitive menu.

   The IF group is no longer visible in the **Outline**, but the section definitions remain.

   All section definition rules, including ELSEIF rules, are reset to IF rules. If a section definition had an ELSE condition, no rule is applied and you will see **Always** next to the definition in the **Summary Info** column.

Moving an IF Group

- To move an IF group, simply drag and drop it from one location in the outline to another.

Note: IF groups cannot be nested. You cannot move an IF group inside of another IF group, or inside of a section definition.

Deleting an IF Group from the Model Document Outline

You can delete a group from the model document outline along with all of the section definitions belonging to it. If you want to remove a group without deleting its section definitions, see *Ungrouping Section Definitions* on page 209.

Method: Remove a group and all of its section definitions

1. Select the group that you want to remove in the **Outline**.
2. Choose **Outline > Remove**, or right-click the group and choose **Remove**.
3. You are asked if you want to remove the item you have selected. Click **Yes**.

   The group and all of the section definitions belonging to it are deleted.

Chapter 6

Charts and Graphs

This chapter describes:

Charts and Graphs

Unlike Microsoft Word or Excel charts and graphs which are created using static data, IStream charts and graphs are created using dynamic data that is incorporated into the document during the document assembly process. This allows you to specify:

- the data to be used in the chart or graph
- a range of graph and chart types
- the titles, legends, size, fonts, colors and other properties

The chart or graph is defined by the author in the model document and is created at generation time as part of the assembly process.

IStream Visualizer is an optional module available with IStream Document Manager.

Note: The data used to create a chart is based on an array of values in the model document.

Charts and Graphs Limitations

Please note the following limitations of charts and graphs:

- You cannot insert charts or graphs in the header or footer of a document, a text box, a header or footer text box, or a footnote. Charts must be inserted in a TEXT/ENDTEXT block. If you are using a TEXT ONLY section, you do not need to repeat the TEXT/ENDTEXT. For more information, see *TEXT* on page 81.
- You can only create pie charts if the data used has all positive or all negative values. If you use data that has both positive and negative values, an error message is displayed in place of the chart. Pie charts can also be created using data with zero values mixed with either only positive or only negative values.
- If you are querying a Microsoft SQL database to populate an array of data that will be used in the chart or graph, note that NULL values will be charted or plotted as zeroes.
- You cannot use the **Edit** function to change a chart or graph.
- You cannot use IStream Customizer or the IStream Author Add-in for Microsoft Word to customize a chart or graph.

Inserting Charts and Graphs

This section describes how to insert a chart or graph into a model document or section.

Warning: Any titles, values, or labels you enter on the **Chart Area**, **X Axis**, or **Y Axis** tabs might not entirely display when the chart or graph is generated, depending on:

- the font size
- the length of the title, value, or label
- the chart or graph's size

Method: Insert charts and graphs

1. Open or create a model document or section where you want to insert a chart or graph.
2. Ensure that all the data you want to chart or graph has been set: see *Using Arrays* on page 147. This data can be in the master section or a sub-section.
3. Place your cursor between any `TEXT/ENDTEXT` paired rules (see *TEXT* on page 81), or any text-only part in the model document or section where you want to insert the chart or graph.
4. On the **Author** toolbar, click the **Insert Chart** button, or choose **Insert Chart** on the **Author** menu**.**

   The **Chart** dialog box is displayed, showing values in some fields according to the defaults defined for the current user. To change these defaults for all documents, see *Editing Charts and Graphs Defaults* on page 32.
5. To change the settings for this model document go to the **General** tab and complete the following fields:
   - **Chart Name** – a mandatory field: enter the name you want to use to identify the chart or graph in the section - it does not have to be a unique name.
   - **Chart Type** – click the drop-down list and select one of the following types:
     - **Bar 2-D** (a two-dimensional bar chart)
     - **Bar 3-D** (a three-dimensional bar chart)
     - **Column 2-D** (a two-dimensional column chart)
     - **Column 3-D** (a three-dimensional column chart)
     - **Line** (a line chart)
     - **Pie 2-D** (a two-dimensional pie chart)
     - **Pie 3-D** (a three-dimensional pie chart)

Note: The **X Axis**, **Y Axis**, or **3-D View** tabs display or remain hidden depending on the chart type you select.

- **Chart Data** – this is a mandatory field: enter the array of data values to be charted or graphed (for more information on arrays, see *Using Arrays* on page 147), or use any list of variables (for example, `<var1>,<var2>,...,`).

  Use one of the following formats:

  `<ArrayName>` – the name of an array variable defined in the model document: use this format to chart or graph numeric values in the entire array for any chart and graph except a pie chart

  `<ArrayName{Row# list/range}:{Column# list/range}>`

  - **ArrayName** is the name of an array variable defined in the model document.
  - The **Row# list/range** is a list and/or range of row numbers in the array containing the values you specifically want to chart or graph.
  - The **Column# list/range** is a list and/or range of column numbers in the array containing the values you specifically want to chart or graph. Use this format to chart or graph select numeric values in an array for any chart and graph except a pie chart.

  Examples:

Value	Result
{2,4,6-9}:{2-3,5-7,9,11}	generates rows 2, 4, 6, 7, 8, 9, and columns 2, 3, 5, 6, 7, 9, 11
{2,4,6-}:{2-3,5-}	generates rows 2, 4, 6, and so on to the last row, and columns 2, 3, 5, and so on to the last column.
{}:{2-3,5-7,9,11}	generates all the rows, and columns 2, 3, 5, 6, 7, 9, 11
{2,4,6-}:{}	generates rows 2, 4, 6, and on, and all the columns

`<VariableName1>,<VariableName2>,<VariableName3>` specify variables defined in the model document whose values you want to chart or graph. Each variable must have a single value. Use this format to chart or graph numeric data variables.

- **By Column** or **By Row** – Select **By Row** if you want the rows to be taken as the series and the columns as the categories. Select **By Column** if you want the columns to be taken as the series and the rows as the categories.

- **Row** or **Column Labels** – These are optional fields. Enter the value you want used to specify the labels associated with the rows or columns of data you want to chart or graph. Use one of the following formats:
  - `<ArrayName>` is the name of an array variable defined in the model document. Use this format when you want all elements of the array to be associated with the rows or columns of the chart data as row or column labels. This array will have only one row.
  - `<ArrayName{Row #/list/range}:{Column #/list/range}>`

    ArrayName is the name of an array variable defined in the model document. Use this format to specify a one-dimensional set of row or column labels, that is, either one row with multiple columns or one column with multiple rows.
    Examples:

Value	Result
{2,4,6-9}:{1}	uses each element in rows 2, 4, 6, 7, 8, 9, of column 1
{2,4,6-}:{2}	uses each element in rows 2, 4, 6, and onwards, of column 2
{}:{3}	uses each element in all the rows, of column 3
{1}:{2,4,6-9}	uses each element in columns 2, 4, 6, 7, 8, 9 of row 1
{2}:{2,4,6-}	uses each element in columns 2, 4, 6, and onwards, or row 2
{3}:{}	uses each element in all the columns of row 3.

  - `'Label1',<VariableName1>,'Label2',<VariableName2>,<VariableName3>,…,`

    The **'Label1'** and **'Label2'** are strings to be associated as labels with the corresponding rows or columns in the chart data. You can use double quotes (" ") instead of single quotes (' '), and single quotes inside a double-quoted string. You can also use double quotes inside a single-quoted string.

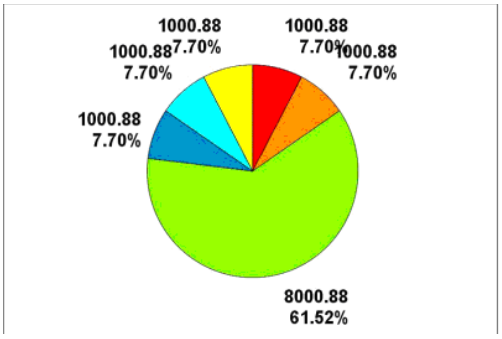
    The **VariableName1**, **VariableName2**, and **VariableName3** are variables defined in the model document containing values to be associated as labels with the corresponding rows or columns in the chart data. Each variable must have a single value.

6. Enter optional values or accept the system default values on the **Chart Area**, **Legend & Data Labels**, **Colors & Patterns**, **X Axis**, **Y Axis**, or **3-D View** tabs, including the following fields:

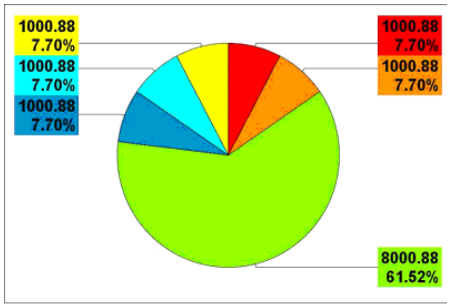
- **Ignore SETFORMAT** on the **Legend & Data Labels** tab. For more information, see *Using SETFORMAT with Charts and Graphs* **on page 216**.
- **Circular Layout** on the **Legend & Data Labels** tab. This option only applies if you selected a pie Chart Type in the General tab.

Using Circular Layout with Pie Charts

If you select the **Circular Layout** check box, labels are placed *around* the chart in generated documents:



If you clear the **Circular Layout** check box, labels are placed on the *sides* of the chart:



7. Click **Apply** to insert the graph or chart and keep the **Chart** dialog box open to further change the graph or chart, or click **OK** to insert the graph or chart and close the **Chart** dialog box. To edit a chart or graph, see *Editing Charts and Graphs* on page 219.

Note: When you select either the **Pie 2-D** or **Pie 3-D** chart types, regardless of how many rows and columns of data you provide in your chart data, only the first row or first column of data (depending on your **By Row** or **By Column** selection) is graphed or charted.

Using SETFORMAT with Charts and Graphs

You can apply SETFORMAT language standards to the numeric values in your charts and graphs in IStream Author, or you can ignore the SETFORMAT function and format chart and graph values according to English standards. If you use SETFORMAT to format numbers in your charts and graphs, you can

customize how decimal points and thousands separators appear for each defined language in the FORMATS table.

Related Topics

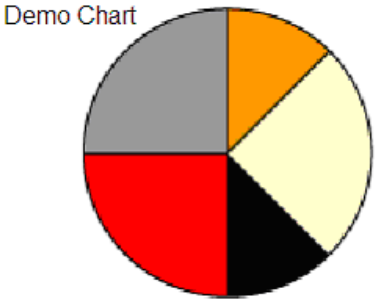
- *SETFORMAT* on page 136
- *Customizing Decimal Separators* on page 137
- *The Ignore SETFORMAT Check Box* on page 217

The **Ignore SETFORMAT** Check Box

When working with charts and graphs, you can choose to **Ignore SETFORMAT** by selecting the check box on the **Legend & Data Labels** tab. By default, **Ignore SETFORMAT** is selected, meaning that values on your chart or graph will be represented according to English standards, even when the SETFORMAT function is in effect throughout the rest of the document. If you want to format chart and graph values according to the language specified by CALL SETFORMAT earlier in the model, clear the **Ignore SETFORMAT** check box.

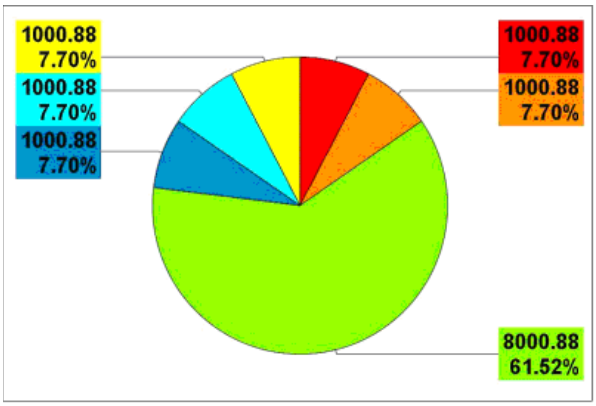
Example: The following shows some sample code and a chart as it would appear in a model document.

```
DEFINE values {}
CALL AADD (values, { 1000.88, 1000.88, 8000.88, 1000.88, 1000.88, 1000.88 } )
CALL SETFORMAT("french")
TEXT
```



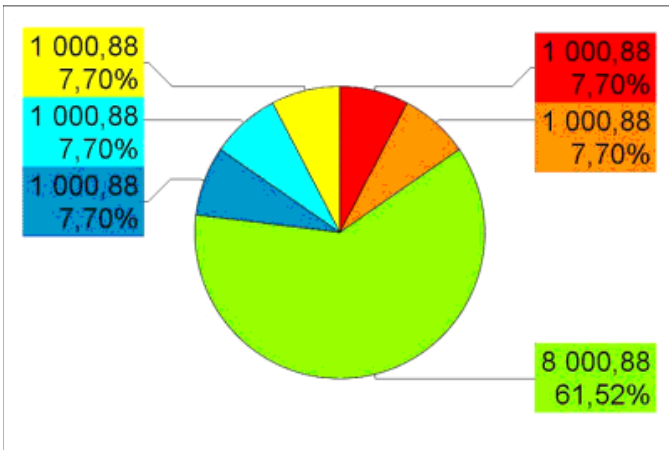

```
ENDTEXT
```

If you select the **Ignore SETFORMAT** check box, your pie chart appears as follows in generated documents:



Even though the SETFORMAT function set the language format for numbers and dates to French, the values in the chart appear in English format.

If you clear the Ignore SETFORMAT check box, your pie chart appears as follows in generated documents:



The values in the pie chart are formatted according to French language standards, using commas (,) in place of decimal points (.) .

Related Topic

- *Customizing Decimal Separators* on page 137

Editing Charts and Graphs

This section describes how to edit charts or graphs in a model document or section.

Warning: Any titles, values, or labels you enter on the **Chart Area**, **X Axis**, or **Y Axis** tabs may not entirely display when the chart or graph is generated, depending on:

- the font size
- the length of the title, value, or label,
- the chart or graph's size

Method: Edit charts and graphs

---

1. Open a model document or section containing the chart or graph you want to edit.
2. Double-click the chart or graph, or right-click the chart or graph and select **Chart Object > Edit**.

   The **Chart** dialog box is displayed, showing the previously saved values entered in each field. If an alert is displayed, make sure your security setting for macros is set to low: select **Tools > Options > Security > Macro Security** to stop the alert from appearing.
3. Change, remove, or add values in any field. For more information see *Insert charts and graphs* on page 213.

Note: When you select either the **Pie 2-D** or **Pie 3-D** chart types, regardless of how many rows and columns of data you provide in your chart data, only the first row or first column of data (depending on your **By Row** or **By Column** selection) is graphed or charted.

Chapter 7

Troubleshooting

This chapter includes information to help you solve problems you may encounter while authoring your model documents and sections.

This chapter describes:

Common Issues

The section you just added to a model document does not appear in the tree view.

There may be two reasons for this:

- You may not have compiled the section to update the tree view. If you added a sub-section to a currently active section using the INCLUDE rule, and then compiled the section, only the part of the tree view related to that active section is refreshed or updated to show the new sub-section. If you added sub-sections to other sections and have not recompiled them, you do not see any changes in the tree view.
- You may need to expand the branch for that section to view the entire tree. Double-click the branch to expand or close it as required.

Try compiling again to see if your sections now appear in the tree view by selecting **Compile Section**, **Compile All Changed**, or **Rebuild Model Document** from the **Author** menu.

A new section you just created in the Editor window does not appear in the tree view.

You must include a reference to that section in the model document before it appears in the tree view and is assembled with the new IStream document during generation.

In IStream Author, you can have multiple sections open in the Editor window, whether they are referenced in the open model document or not. The number, content, and position of the IStream Author windows for that session is saved when you close a model document.

When trying to create a new section using the “Based On” option for a Word model document file, an error message indicates that the data may be corrupt because of “improper WP storage”

You chose the wrong format for creating the new section. Use the **Import** option to apply an existing Word DOC file as the basis for your new section. The **Based On** option is only for using existing sections or master sections (CDS or CMS) files as initial content for your section.

You do not see any TEXT...ENDTEXT rules in the section you are reviewing, but it still generates correctly.

The section is probably a **Text Only** section. To check this, select **File > Section Properties** in IStream Author. If there is a check mark beside **Text Only**, it means that this section is being evaluated as a complete block of text and does not required the TEXT...ENDTEXT rules.

Alternatively, there may indeed be a TEXT...ENDTEXT pairing surrounding your text, but it might be hidden. In the **Tools** > **Options** menu, click the **View** tab, and see if there is a check mark next to **Hidden Text** in the **Non-printing Characters** section of the tab. If there is no check mark, it means that there may be hidden text

in your section, but it is not visible. Click the check box to **View Hidden Text**, and then examine your document.

Rules in a section are printing as text.

The rules may be enclosed in a TEXT...ENDTEXT block, or the section may be tagged as **Text Only**. Check that your rules are outside any TEXT...ENDTEXT pairing. Then, select **File > Section Properties** and verify that there is no check mark next to the **Text Only** option.

A newly-configured InfoSource does not appear in the list when browsing IStream Author's Options.

Ensure that IStream Author is closed when configuring the InfoSource. If IStream Author is open when you use InfoConnector to configure an InfoSource, IStream Author does not recognize the new InfoSource until you close IStream Author and then reopen it.

When trying to open a document, IStream Author indicates that it is "locked for edit".

Someone else may be using the document. Wait for them to finish their work, then try opening the document again.

The document may not have been properly saved in IStream Document Manager, or the generation of the document failed. If so:

a. Navigate to the document in IStream Document Manager.

b. Reset the Reserve on the document.

c. Open the model document and regenerate the document.

Content that appears in the header and footer of the model document is not appearing in generated .clg documents.

If you do not include any section breaks in your model document, content in the header and footer will not appear in your generated `.clg` documents. Insert a section break into the model document, then generate the document again.

Common Questions

How do I get the content of an IStream document into a PDF file?

In Word, click **Author > Export CLG to PDF...** , or choose  on the toolbar.You can choose to save the PDF anywhere on the file system. Refer to *Hyperlinks and PDF Bookmarks* on page 171 for more information on creating and preserving hyperlinks and PDF bookmarks. See also *Model Document Considerations* on page 165 for considerations regarding Word's Track Changes feature and how it affects exporting to PDF.

How can I see all my sections open at the same time in one window?

To view all your sections at one time in the same window, follow these steps:

- Close the **Outline** and **Log** windows and remove the toolbars to give you more space on your screen for the **Editor** window.
- Choose **Open Section** from the **File** menu to open each section you want to view. If there are more than 9, they will overlap and you cannot select them from the **Window** menu without going to an additional dialog box.)

Complete any of the following steps to view multiple sections:

- Choose **Cascade** from the **Window** menu to display the windows one in front of the other.
- Choose **Tile Horizontally** (or **Tile Vertically**) from the **Window** menu to see each window in reduced size, and side by side.
- Experiment with the position of the windows and toolbars. You can always open the windows and toolbars again if you close them. Note that the final window and tabular configuration displayed on your screen is saved when you close IStream Author.

Error Messages

The following table explains IStream Author error messages that you may encounter while creating and generating model documents. Descriptions and possible solutions are included. Some messages involve the use of the authoring language, InfoSources, or other aspects of the program.

These messages can appear in the compile log, the generation log, or onscreen. If you cannot resolve the problem, contact your system administrator. If the problem persists, contact Customer Support.

Message	Description
Cannot add document.	The repository InfoSource cannot add the IStream document specified in the GENERATE rule because the repository is full or the document reference is incorrect.
Cannot connect to InfoSource.	The InfoSource exists, but it cannot access its data because: • the InfoSource is incorrectly configured • there is a missing DSN used by the ODBC or USERDB InfoSource • the authentication is incorrect Check the user ID and password and the InfoSource configuration.
Cannot execute application.	The application specified in the APP function failed to execute because. • the specified path points to an invalid or missing application • the specified string parameter is illegal
Cannot get variable from InfoSource.	The specified data item is not provided by the InfoSource because the specified database field section or file does not exist. Check the validity of the references to the database fields and files.
Cannot load document.	The model document failed to load because it is missing or corrupt.
Cannot open compound file.	A section or IStream document file failed to open because of a missing section or IStream document file, or a corrupt file. Check that the file exists.
Cannot open document.	The repository InfoSource cannot open the existing document in the GENERATE rule because: • the InfoSource configuration is invalid • the IStream document was reserved by another application. Check the InfoSource configuration or unreserve the IStream document.

Message	Description
Cannot save IStream document.	The Repository InfoSource cannot save IStream document in the GENERATE rule because of a repository failure or incorrect InfoSource configuration.
Dump variable.	Subsequent lines contain information about the data type and value of the specified variable. This is not an error, but is simply the output of the DISPVAR function: no action is required.
Extra ENDTEXT/ENDTEXT+/ENDIF/ ENDDO/NEXT/UNUSEIS	There is an unmatched ENDTEXT, ENDTEXT+, ENDIF, ENDDO, NEXT, or UNUSEIS rule at the end of the section.
Failed to copy files.	You tried to save a model document or section with a filename longer than eight characters. The InfoSource you are attempting to save to (such as a network drive) may not support long filenames. Shorten the filename or choose a different repository.
Failure evaluating expression.	The model document or subsection reference could not be evaluated because: • there is an invalid InfoSource or section reference in an INCLUDE rule • the model document reference for generation is invalid Check the validity of InfoSources and section references in the INCLUDE rules.
General syntax error.	There is an unrecognized syntax error in an expression. This is caused by any type of wrong syntax. Validate the expression syntax.
Generation date does not fall in the effective and termination date range.	The section or subsection cannot be generated because the effective date, or termination date of the section or subsection falls outside what was set for generation.
Illegal IStream document reference.	The IStream document reference in the GENERATE rule is invalid because: • the IStream document cannot be properly accessed • the repository does not contain the path
Illegal parameter value in function call.	A function cannot accept a value in one or more of its parameters because: • the numeric value is out of range • a string value is in an invalid format (for example, in the CTOD function)

Message	Description
InfoSource does not exist.	An InfoSource specified in an expression or in the GENERATE rule is not defined because: • its name is misspelled • its name is correct, but it is not defined in InfoConnector
InfoSource System is expected in checkup function.	The default "System" InfoSource does not maintain the table specified in a CHECKUP function because: • the table name is misspelled in the function • the table name is correct, but is maintained by an InfoSource other than the "System".
InfoSource name is expected in lookup function.	The default "System" InfoSource does not maintain the table specified in a LOOKUP function because: • the table name is misspelled in the function • the table name is correct, but is maintained by an InfoSource other than the "System"
InfoSource name is expected in query rule.	A SQL statement in a QUERY rule query cannot be run in any default InfoSource because: • the InfoSource name is missing in the QUERY rule • the SQL statement is misspelled • the specified tables are not maintained by any default InfoSource
InfoSource name is expected in SELECT rule.	No default InfoSource can SELECT the specified table because: • no default Userdb InfoSource contains the specified table in its mapping • the InfoSource name is missing in the SELECT rule Validate the mappings of the Userdb InfoSources, or add the missing InfoSource name parameter.
Invalid array index.	The array index is invalid because a non-existing array item is accessed in an expression, or is changed by a DEFINE rule. Check the logic of the model document.
Invalid InfoSource is specified in checkup function.	An InfoSource provided in a CHECKUP function does not maintain a specified table because the InfoSource or table name is wrong.
Invalid InfoSource is specified in lookup function.	An InfoSource provided in a LOOKUP function does not maintain a specified table because the InfoSource name or table name is wrong.
Invalid InfoSource is specified in query rule.	An InfoSource provided in a QUERY rule cannot execute a specified SQL statement because: • the InfoSource name is wrong • SQL statement is misspelled

Message	Description
Invalid InfoSource is specified in SELECT rule.	The Userdb InfoSource provided in a SELECT rule does not support the specified table because: • the InfoSource name is wrong • there is a missing mapping in the Userdb InfoSource
IStream document reference is missing in rule GENERATE.	A GENERATE rule contains an empty or missing reference to an IStream document.
Keyword BEGIN is expected.	An END rule does not have a corresponding BEGIN rule.
Keyword END is expected.	A BEGIN rule is missing a corresponding END rule.
Logical expression is expected.	An expression in an IF or DO WHILE rule is not logical. The rule acts as if the expression was evaluated to False. Correct the preceding errors then verify the logic of the model document.
Maximum level of recursive include is exceeded.	The maximum nesting level of included (recursive or not recursive) sections has been exceed, caused by: • an infinite recursion • too many nested INCLUDE rules The INCLUDE rule will therefore not be performed. Change the logic of the model document to prevent infinite recursion or reduce the nesting of INCLUDEd sections.
Missing ELSE, ELSEIF or ENDIF for IF	An IF rule is missing a a corresponding ELSE, ELSEIF or ENDIF rule. (IF...ELSE...ENDIF or IF...ENDIF)
Missing ENDDO for DO WHILE	A DO WHILE rule is missing a corresponding ENDDO rule.
Missing ENDIF for IF	An IF rule followed by an ELSE rule is missing a corresponding ENDIF rule.
Missing ENDTEXT for TEXT	A TEXT rule is missing a corresponding ENDTEXT rule.
Missing expression in DO WHILE rule.	A DO WHILE rule is missing an expression.
Missing expression in IF rule.	A IF rule is missing an expression.
Missing NEXT for INITIALIZE	An INITIALIZE rule is missing a corresponding NEXT rule.
Missing NEXT for QUERY	A QUERY rule is missing a corresponding NEXT rule.
Missing query in QUERY rule.	A QUERY rule is missing a SQL statement.
Missing table name in INITIALIZE rule.	An INITIALIZE rule is missing a table name.

Message	Description
Missing table name in SELECT rule.	A SELECT rule is missing a table name.
Missing UNUSEIS for USEIS	A USEIS rule is missing a corresponding UNUSEIS rule.
Model document reference is missing in rule GENERATE.	A GENERATE rule is missing the reference to the model document to be generated (the second parameter).
Must not generate a Sub Section.	A user tried to start generation from a subsection. Generation must always start from master section. Verify the GENERATE rule. Ensure that no subsection is specified in the GENERATE rule.
No InfoSource can initialize table.	Neither of the default InfoSources can initialize the table specified in an INITIALIZE rule because: • the mappings of the default Userdb InfoSources do not contain the specified table • another table was selected in the preceding SELECT rule
Not all Key Data are known.	Not all key data items of the model document were defined before the generation. This warning may not affect the generation results, but you should still provide all the key data values.
Section must have the same word processor as model document.	A subsection was created in different word processor from the master section. Recreate the same section using the same word processor as the master section.
Specified InfoSource cannot initialize table.	A Userdb InfoSource provided in an INITIALIZE rule cannot activate the specified table because: • the provided InfoSource is Userdb • the provided InfoSource is Userdb, but does not contain the specified table in its mapping. Check the InfoSource type, and verify the InfoSource mapping.
Syntax error—empty expression.	The expression passed to evaluation does not have tokens because: • there is empty placeholder in the text • there is an empty expression in a rule • the IStream document file is corrupted Check the syntax in the section, and contact technical support if necessary.
Syntax error—missing right parenthesis.	An expression has a left parenthesis without a corresponding right parenthesis.
Syntax error—missing tokens.	Tokens such as + - : are missing in the expression.

Message	Description
The document is empty. The process to generate the document was not completed properly. Please check the model document. One reason could be that the document is not effective yet, or the model document is being opened by another application.	You have tried to regenerate an existing model document from a model document other than the one loaded in IStream Author. Ensure that you have the correct model document open.
The expression is too complex.	The limit of expression complexity has been exceeded because of too many: • nested expressions, • indexes in an array reference • function arguments Split the expression into several simple ones.
The file is in use by another application.	A section file is open for writing by another application. Close the section file in IStream Author.
Unevaluated Expression <variablename>	This error message appears in your assembled document. It indicates that Assembler was unable to obtain a value for the variable you indicated. Causes include: • if this is a defined variable, the spelling may be incorrect. • you may have included a $ when you defined the variable but not when you included it in the document • the variable has no corresponding field in the database, or its syntax may be incorrect
Unknown function.	There is an unknown function in the expression. This can be caused by: • a misspelled function name • the function is actually a variable: a token (+ -) is missing between it and the subsequent parenthesis
Unknown rule - cannot compile section.	There is an illegal rule keyword. It may be misspelled, certain rule text may be missing or the rule nesting may be wrong.
User generated error.	The DOCERR function was executed. Debug the model document.
Variable is not defined.	A variable used in an expression or placeholder has not been defined. Check the logic of the model document, and resolve any other errors.
Variable must not be defined.	You have tried to define a variable using a reserved word, such as LANGUAGE. You cannot use reserved words as variable names.

Message	Description
Wrong number of parameters in function call.	A CALL function does not accept this number of parameters. Verify the expression syntax and lower the number of parameters.
Wrong type of parameter(s) in function call.	A CALL function does not accept this type of parameter. Verify the model document logic.

INDEX

D

G

H

I

T