

IBM WebSphere Transformation Extender



Map Designer

Version 8.1

Note

Before using this information, be sure to read the general information in "Notices" on page 181.

October 2006

This edition of this document applies to IBM WebSphere Transformation Extender Version 8.1; and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this document, email DTX_doc_feedback@us.ibm.com. We look forward to hearing from you.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 2006. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Chapter 1. Introduction to the Map Designer	1
Using the Map Designer	1
Building the desired output data	1
Define data objects and properties	1
Create maps to specify sources and targets	2
Chapter 2. User interface	3
Navigator window	3
Navigator icons	3
Navigator > List view	4
Navigator > Composition view	4
Viewing the Navigator	4
Showing the navigator	4
Hiding the Navigator	4
Navigator docking display option	5
From window	5
Functions and features of the From window	5
To window	5
Functions and features of the To window	6
Resize the Output and Rule columns	6
To window shortcut menus	6
Map card context menus	6
Floating additional cards in the main window	7
Viewing choice groups in the To window	7
Organizer window	7
To view the Organizer window	7
To float Organizer window tabs	8
Unresolved rules tab.	8
Remarks tab	8
Trace tab.	8
Audit an object	8
Audit log	8
Build results	8
Rule Bar	8
Rule bar docking display option.	9
Showing and hiding the rule bar	10
Movable Windows	10
Map Designer toolbar	10
Title bar	10
Status bar	10
Menu commands and tools	11
Map Designer menus	11
Chapter 3. Map Designer basics	17
Map Designer file name extensions	17
Starting the Map Designer	17
Configuring the Map Designer environment	18
General options	18
From window options.	18
To window options.	19
Rule Bar options.	19
Navigator	20
Confirmations	20
Organizer window options	20
Run results	21

Run options	21
Properties window options	22
Referenced Maps option	22
Profiler options	22
Debug options	23
Search facilities in the Map Designer	23
Using the find operation	23
Using a replace operation.	24
Using a more find/replace operation	24
Printing	25
Printed map details.	25
Printing map details	26
Printing map settings	26
Printing input and output card information	26
Printing Organizer information.	27
Printing run results.	27
Chapter 4. Maps and map source files	29
Creating a map source file	29
Creating a map	30
Map name guidelines	30
Map source file differences	30
Comparing map source files.	31
Comparing objects with different names.	31
Viewing map source file differences	31
Resizing the map source file differences window	32
Map differences	32
Chapter 5. Input and output cards	35
Card overview	35
Card specifics	35
Viewing input and output cards	36
Compositional hierarchy	36
Creating input and output cards	36
To create an input card	36
To create an output card	37
Editing an input or output card	37
Chapter 6. Card settings	39
Schema	39
CardName.	39
TypeTree	39
TypeName.	40
SourceRule input card settings	40
SourceRule Setting overrides	40
FetchAs	41
FetchAs > WorkArea	42
FetchAs > FetchUnit	42
GET	44
Source	44
Source > FilePath	44
Source > Command	44
DatabaseQueryFile	44
DatabaseQueryFile > Database	45
DatabaseQueryFile > Query	45
DatabaseQueryFile > File.	45
Transaction	45
OnSuccess	45
OnFailure	45
Scope	46

TargetRule output card settings	47
TargetRule Setting overrides	47
PUT	48
FilePath	48
Target > Command	48
DatabaseQueryFile	48
DatabaseQueryFile > File	48
DatabaseQueryFile > Database	48
DatabaseQueryFile > Table	48
Transaction	48
OnSuccess	48
OnFailure	49
Scope	49
Input and output card settings	50
Retry	50
Retry > Switch	50
Retry > MaxAttempts	50
Retry > Interval	50
DocumentVerification	51
Backup settings	52
Defining backup settings	52
Backup > Switch	53
Backup > When	53
Backup > BackupLocation	53
BackupLocation > Directory	53
Directory > Value	53
BackupLocation > Filename	53
BackupLocation > FileName > Action	54
BackupLocation > FileName > Value	54

Chapter 7. Map rules. 55

Map rules overview	55
Color coding of map rules	55
Enabling color coding for map rules	56
Configuring colors for map rules	56
Map rules as expressions	56
Map rules define how to generate data objects.	57
Map rules are evaluated independently in sequential order	58
Unavailable rule cells	58
Map rule for the entire output object	58
Map rule for each component	58
Entering map rules	59
Inserting a function into a map rule	59
Inserting a symbol into a map rule	59
Entering a restriction into a map rule.	60
Generating no output	60
Special characters in map rules	60
Referencing an output in a map rule	60
Comments in map rules	61
Syntax errors in map rules	61
Data object names in map rules	61
Card names in map rules.	62
Component names in map rules	62
Index in map rules	63
Partition in map rules	63
Use ellipses	63
Using ellipses to shorten an object name.	63
Viewing properties in the Map Designer.	64
Viewing type properties	64
Viewing component rules.	64
Viewing component attributes	65

Viewing restrictions of an item	65
Viewing function syntax	65
Viewing map rule properties	65
Viewing functional map inputs and outputs	66
Opening the type tree	66
Formatting a map rule.	66
To create a new line in the rule bar if Enter is the Commit key	66
To create a new line in the rule bar if Tab is the Commit key	66
Deleting the current selection in map rules	67
Chapter 8. Formulating map rules	69
Generating no output	69
Text literals as NONE	69
When input data is missing	69
When portions of an input series are missing	69
When an output evaluates to NONE	69
Mapping an input item to an output item	71
Automatic item conversions	71
Intermediate item conversions	71
Concatenating text strings	72
Mapping a group to a group	72
Automatic conversion of syntax items	72
Mapping to a group with a maximum range of 1.	72
Outputs with maximum range greater than 1	72
Indexing an output	73
Reordering indexed objects	73
Mapping to multiple occurrences of a group	73
Indexing an Input	74
Mapping from a floating component type	74
Mapping from a floating component of a particular component	74
Chapter 9. Functional map basics	77
When to use a functional map	77
Similarities of input and output data	77
Using a functional map	78
Example scenario for using a functional map	78
Comparing executable and functional maps	79
Syntax of a functional map expression	80
Determining the arguments of a functional map	81
Entering the map rule that references the functional map	81
Creating a functional map	82
To create a functional map	82
To create input and output cards in the functional map.	82
Input card types of a functional map	83
Input arguments as data objects	83
Output card type of a functional map	84
Entering map rules	84
Using multiple inputs in a functional map	84
Input objects are triggers	85
Going to a functional map referenced in a map rule.	85
Functional maps in the Navigator	85
Functional maps that do not exist	85
Dragging a functional map name into a rule	85
Chapter 10. Functional Map Wizard	87
Using the Functional Map Wizard	87
Properties of the functional map	87
Map name.	87
Input cards	87
Output card	88

Referencing a map from the Functional Map Wizard	88
Editing card definitions in the Functional Map Wizard	89
Resetting the Functional Map Wizard.	89
Undefined card and maps	89
When an argument is invalid	90
When an argument returns an object of a known type	90
Creating more than one functional map	90
Selecting multiple map rules.	90
If a map already exists	91
Using an output as an argument	91
Multiple levels of functional maps.	91
Using a functional map to increment data	92
Referring to the current output in a map rule	92
Using a functional map to reorder data	92
Chapter 11. Map settings	95
Defining map settings for a single map	95
Defining default map settings	95
MapAudit settings	96
MapAudit > Switch	96
MapAudit > BurstAudit	96
BurstAudit > Data	96
BurstAudit > Execution	97
MapAudit > SummaryAudit.	98
SummaryAudit > Execution	98
SettingsAudit.	98
SettingsAudit > Data	98
SettingsAudit > Map	99
MapAudit > AuditLocation	99
Directory	99
FileName.	100
Value	100
FileName > Action	100
FileName > Value	101
AuditLocation > Sized	101
MapTrace settings	101
ContentTrace	101
ContentTrace > Switch	101
ContentTrace > TraceLocation	101
Directory	101
Value	102
FileName.	102
Value	102
InputContentTrace.	102
CardNumber	102
InputContentTrace > StartObject	102
InputContentTrace > EndObject	102
RulesTrace	103
CardNumber	103
SummaryContentTrace	103
WorkSpace settings	103
PageSize	104
PageCount	104
Directory	104
WorkFilePrefix	104
Century settings	105
Century	105
Century > CCLookup	105
Validation settings.	106
Validation	106
OnValidationError.	106

RestrictionError	106
SizeError	107
PresentationError	107
Retry settings	107
Retry > Switch	107
Retry > Interval	108
Retry > MaxAttempts	108
Warnings settings	108
Warnings > Return	109

Chapter 12. Configuring bursts 111

Batching logical messages	111
Using bursts for large files	111
Configuring bursts for error recovery	112
Configuring the RUN function and bursts	113
Logical or adapter-specified FetchUnits	113

Chapter 13. Managing maps 115

Standard windows capabilities	115
Moving and copying objects by dragging	115
Selecting objects	115
Deletion	115
Objects that can be deleted with the delete key	115
Cut, copy, and paste	115
Objects that can be moved by dragging	116
Objects that can be copied by dragging	116
Additional mouse shortcuts	116
Moving a map	116
Copying a map	116
To copy a map by dragging	117
To copy a map using the copy command	117
Deleting a map	117
To delete a map using the Delete key	117
To delete a map using the delete command	117
Merging a map	117
To merge a map by dragging	117
To merge a map using the Merge command	118
Renaming a map	118
Exporting a map source file	118
XML format for maps and map rules	118
Exported map settings	119
Exported card settings	119
Exported map rules	120
Referencing maps	120
Version control	120
How to reference a map	121
Using a resource alias with referenced maps	121

Chapter 14. Managing cards 123

Moving a card	123
Converting a card	123
Copying a card	123
To copy a card by dragging	123
To copy a card using the copy command	123
Deleting a card	124
To delete a card using the Delete key	124
To delete a card using the delete command	124
Editing a card	124
Editing a card	124
Changing a card name	124

Changing the type of an input card	125
Changing the type of an output card	125
Changing the type tree of a card	125
Effects of modifying a type tree	126
Changing a type that is used for a card.	126
Rebuilding a map that has changed	126
Chapter 15. Building maps.	127
Referenced maps	127
Building a map.	127
Using the Organizer window	128
Viewing and using unresolved map rules	128
Remarks	129
Data audit settings	129
Trace tab	129
Type tree errors	130
Map build analysis errors	130
Viewing build results.	131
Searching in the Organizer	131
Map build warning messages	131
Map build compile errors	132
Viewing build results.	132
Compiled map files	132
Building a map for non-windows platforms	132
Platform-specific file name extensions	133
Chapter 16. Running a map	135
Running a map.	135
Map error and warning messages	135
Run results	135
Viewing the map run results	136
Closing Run Results windows.	136
Viewing run results with shortcut access methods	137
Chapter 17. Map performance	139
Map Profiler overview	139
Map Profiler output	139
Type names	139
Function times	139
RUN maps	139
Configuring the Map Profiler	139
Command line	140
Using the Map Profiler	140
Profiler output example	140
Profiler summary report example.	141
For best results...	142
Chapter 18. Managing invalid data	143
Using the Restart Attribute	143
Using the REJECT function.	143
Using the ISERROR function	143
Using the CONTAINSERRORS function	144
Chapter 19. Map audit overview	145
Audit log contents.	145
Settings that control audit log content	145
Viewing the audit log	145
Execution summary in the audit log.	145
Source data report by card	146
Target data report by card	146

BurstAudit Data in the audit log	148
Data content information in the audit log	148
Auditing data content	149
Data settings in the audit log	149
Map settings in the audit log	150
Auditing a map	150
Mapping from the audit log	151
ExecutionSummary in the Tryaudit type tree	151
Configuring data audit	151
Status codes in the audit log	153
Chapter 20. Troubleshooting	155
Overview of debugging aids	155
Command Server window	155
Run results	156
Audit log	156
Customized validation settings	156
Map Debugger	156
Breakpoints	156
Rule execution	157
Viewing object values at a rule level.	157
Running the Map Debugger	158
Remote debugging	158
Trace files	159
Trace for general data	160
Trace for XML data	165
Chapter 21. Error and warning messages	167
Map execution error and warning messages	167
Map execution warning messages	172
Map build error messages	172
Map build warning messages	176
Map compile error messages	176
Audit log status codes	178
Valid audit log status codes	178
Warning audit log status codes	178
Error audit log status codes	178
Notices	181
Programming interface information	183
Trademarks and service marks	183
Index	185

Chapter 1. Introduction to the Map Designer

The Map Designer is a client component of the Design Studio that you use to develop maps that define input and output specifications and mapping rules for data transformation.

The Map Designer uses the definitions of data stored in the type trees (that are created using the Type Designer) to specify the transformation logic in the form of map rules. Map rules operate on input data objects and build output data objects. The map can be built for specific platforms, and then run on that platform to perform the transformation of the data.

During the design phase, you build and run maps from within the Map Designer on a Windows platform. You can view the results of a map in the Map Designer. You can also use the Map Designer to build a map to be run on another platform.

Using the Map Designer

In order to use the Map Designer, you must already have the type tree(s) that define your data. The Map Designer uses the data object definitions that are stored in those type trees.

The Map Designer is used to:

- Create maps to specify the logic necessary to transform the input data to the desired output data.
- Identify the source and data objects of the input data.
- Validate and resolve the source data type properties defined in the Type Designer.
- Identify the target and data objects of the output data.
- Specify and build the output data according to the map rules.
- Provide information about data validation by generating trace files.
- View the run results of the map execution.

After defining data objects and their properties in the Type Designer, you define a map in the Map Designer where map cards specify the input source and the output target.

Building the desired output data

Building the desired output data consists of two main steps:

- Defining the data definitions in the Type Designer.
- Creating maps in the Map Designer.

Define data objects and properties

The type trees are maintained in the Type Designer, although type trees can also be created using the Type Tree Maker, the Importer Wizard in the Type Designer, and the Database Interface Designer.

To define data definitions:

1. Use the Type Designer, Type Tree Maker, or Importer Wizard to create or generate a type tree file. The file name extension of type tree files is .mtt.
2. Define the type properties of input and output data.
3. Specify restrictions of input data for item types.
4. Specify desired validation of input data in component rules.
5. Save and analyze the type tree file (.mtt).

Create maps to specify sources and targets

After defining data objects and their properties in the Type Designer, define the map in the Map Designer where you specify the source of the input and the target of the output. Sources and targets are specified with map cards. Each map can have none or more input cards, and one or more output cards.

To create maps using the Map Designer:

1. Create a map source file. The file name extension of map source files is .mms.
2. Create required executable map(s) in the map source file (.mms).
3. Create an input card that represents your input data.
4. Create an output card that represents your desired output.
5. In the output card map rules, specify the logic necessary to transform the input data to the desired output data.
6. Build the map. This creates a compiled map file (.mmc).
7. Run the map.

Chapter 2. User interface

The Map Designer window includes a Navigator, **Organizer**, title bar, menu bar, toolbar, rule bar, and status bar that provide a graphical user interface to create and manage maps, run maps, and view map results.

The map source files shown in the **Navigator** contain maps that include input cards (in the From window) and output cards (in the To window).

When you create a map source file in the Map Designer, the Save As dialog box is displayed. You can change the default file name **MapSourceFile** (followed by a sequential number) and location. After a map source file is created, the name of the map source file appears in the Navigator indicating that you can begin creating maps, input cards, and output cards. Save the map source file with a filename and a file name extension of .mms.

Navigator window







The Navigator window displays a list of open map source files, the maps contained in those files, and the cards contained in those maps. To view the Navigator if not present, select **View** → **Navigator**.

The Navigator icons graphically represent all of your opened map source files and the maps that they contain. The **Navigator** provides a graphical representation of the input and output cards in the maps.

The Navigator provides two views: List view and Composition view. Select your view option by clicking either tab at the bottom of the Navigator.

Navigator icons

In addition to the alphabetical and compositional structure showing the contents of each map source file, information about entries in the Navigator is conveyed visually with icons.

Icon	Description
	map source file
	map
	functional map
	reference map
	input card
	output card

Navigator > List view

The List view presents a detailed list of the open map source files, the maps contained in the map source file, and the input and output cards for each map. The map source files, maps, and cards are displayed alphabetically from top to bottom.

Each map source file can be expanded to view all of the maps it contains. Each map can be expanded to view the input and output cards. The input and output card labels may also be expanded to view all of the input and output cards. The card number appears in front of the card name.

- Double-click any executable map to display the map in the main window.
- Double-click any input or output card to view that card in the main window.

In the following List view, **ContactFile** is input card #1 for the **AddressToMailingFile** map and **LabelFile** is output card #1.

Navigator > Composition view

The Composition view of the Navigator shows how maps are related. The Composition view displays the map source files and maps arranged in a compositional hierarchy. Functional maps appear underneath the executable map that calls them.

Notice that the maps in Composition view are not alphabetically arranged (as they are in List view), but are arranged in a compositional hierarchy.

Viewing the Navigator

The Navigator can be shown or hidden with the Navigator command on the **View** menu or from the context menu (as shown below). The Navigator can be presented as a docked window or a floating window. Choose the display option from the context menu.

To access the context menu for the Navigator commands:

Right-click the top border.

Showing the navigator

The Navigator can be shown or hidden using the Navigator command on the **View** menu or from the **Navigator's** context menu.

To show the Navigator:

From the **View** menu, select **Navigator**.

A check mark appears next to Navigator in the **View** menu indicating that the Navigator is displayed.

Hiding the Navigator

The Navigator can be shown or hidden using the Navigator command on the **View** menu or from the **Navigator's** context menu.

To hide the Navigator:

To hide the Navigator when it is displayed, select **Hide** from the context menu of the Navigator or from the **View** menu, select **Navigator** to clear the Navigator selection from the **View** menu.

Navigator docking display option

The Navigator can be presented as a docked window or a floating window. To access the context menu, right-click on the **Navigator's** top border.

To dock or float the Navigator window:

1. Right-click on the **Navigator's** top border.
2. From the context menu, select **Allow Docking**.

If a check mark is displayed next to the **Allow Docking** option in the context menu, the Navigator window can be docked.

After selecting **Allow Docking**, you can toggle between a docked window and a floating window by double-clicking the top border of the Navigator.

3. Double-click in the area beneath the title bar so that the Navigator is again docked.

From window

The From window of the Map Designer represents the input data or the data from which you are mapping. The From window contains none or more map cards. Map cards in the **From** window of the Map Designer represent source data and are called map input cards. The From window contains an input card for each data object from which you are mapping.

The cards in the From window display the composition of the data defined in the Type Designer.

Functions and features of the From window

- Edit the input card by double-clicking the title bar of the card (not the window). The title bar of the card displays the card number, card name, and type name.
- Copy an object name from an input card in the From window to an output card in the To window by dragging the object to the rule cell on the To window.
- Insert an object name in a map rule by dragging it from the From window to the rule bar.
- For maps with more than one input card, you can float additional cards in the main window to view more than one card at a time.

To window

The To window of the Map Designer represents the output data to which you are mapping. The To window contains one or more map cards. Map cards in the **To** window of the Map Designer represent target data and are called output cards. The To window contains an output card for each data object to which you are mapping.

The cards in the To window have two columns: the **Output** column and the **Rule** column. The **Output** column lists the name of each output data object. The **Output**

column shows the compositional hierarchy of the output data objects. The **Rule** column contains the mapping rules that provide the logic for building each object in the output.

Functions and features of the To window

The To window contains the output cards that contain the data objects to which you are mapping. The following functions and features enable you to access the commands relevant to the To window.

- Edit an output card by double-clicking the title bar of the card (not the window). The title bar of the card displays the card number, card name, and type name.
- Insert an object name by dragging it from a card to the rule bar.
- Float additional output cards in the main window to view different parts of the same card, or two or three different cards. This enables you to:
 - View multiple input or multiple output cards of a map at the same time.
 - Drag a map rule from one card to another.
To map an object from card #1 to card #2, display both cards by dragging one card to the main window. Then drag the data object from card #1 into a rule cell of card #2.
- Find the active field in the To window by viewing the grayed crosshatch pattern.
- Resize the **Output** and **Rule** columns.
- Access shortcut menus

Note: You cannot drag an objects to the rule cell on the output card. You must drag the data object to the rule bar instead.

Resize the Output and Rule columns

The **Output** and **Rule** columns on the To window may be resized using the splitter cursor.

To resize the output or rule columns in the To window:

1. Position the cursor as shown between the top of the **Output** and **Rule** columns.
2. When splitter cursor appears, drag column to desired size.

To window shortcut menus

Shortcut menus on the output card provide access to commands for data objects. Three areas of the output card on the To window provide access to the following shortcut menus:

- Right-click the title bar of the card.
- Right-click the data object name in the **Output** column.
- Right-click the map rule in the **Rule** column.

Map card context menus

Map Designer context menus are available from the To or From windows by right-clicking in certain areas. These menus offer access for commands for data objects on the To and From windows, and the rule cell on the To window.

Floating additional cards in the main window

If the selected map has more than one card displayed in the To window or From window of the Map Designer, cards may be displayed simultaneously by using the float window feature.

To float additional cards in the main window:

1. Display the cards for the selected map by double-clicking the map in the Navigator.
2. Drag a card from the To or From window into the main window.
Floated cards are view-only. To edit a map rule on an output card, select that card in the To window, not the floated window.

Tabs in the Organizer window can also be floated in separate windows.

Viewing choice groups in the To window

For choice group components that have no rules, but have subcomponents with rules, the component's rule field appears with a grid pattern.

Expand the component to view the rule fields of the subcomponents.

Organizer window

The Organizer window contains information about the selected map. The information includes unresolved rules, remarks, audit settings, the trace file, the audit log, and build errors.

To view the Organizer window

1. In the Navigator, select the map.
2. From the **Map** menu, choose **Organizer**.

Each Organizer window displays information about the selected map. Multiple Organizer windows for the same map, and for different maps, may be open at the same time.

The Organizer tabs include:

- Unresolved Rules
- Remarks
- Data Audit Settings
- Trace
- Audit Log
- Build Results
- Profiler

When you select the **Unique** filename option for the audit log (**MapAudit** → **AuditLocation** → **FileName**), the audit log is not be viewable from the Organizer.

Each tab on the Organizer window can be floated as a separate window. The font of each tab can be customized.

To float Organizer window tabs

Right-click any tab and choose **Float In Main Window**.

Unresolved rules tab

Unresolved map rules appear on the **Unresolved Rules** tab on the **Organizer** window.

Remarks tab

You can enter general comments or comments for particular data objects in a map by using the **Remarks** tab in the Organizer window.

Trace tab

A map can be configured to produce a trace file.

Use the **Trace** tab on the Organizer window to view the trace file.

Audit an object

You can specify that you want to audit a particular object by including that object in the Organizer window on the **Data Audit Settings** tab.

Audit log

A map can be configured to produce a map audit log, which consists of execution statistics, data content information, map settings, and data settings. To view the audit log for the currently selected map, click the **Audit Log** tab on the Organizer window.

To configure an audit log specify the **MapAudit** map settings and any specific data objects to be included in the audit.

If the map setting value for the **AuditLocation Filename = Unique**, the audit log cannot be viewed from the Organizer window.

Build results

Errors and warnings are reported on the **Build Results** tab of the Organizer window. Map build results provide information on errors and warnings that occur during the map build process.

The **Organizer** can be configured to automatically open if there were errors when a map is built. Enable the **Tools Options** → **Organizer** option for **Show only on errors**.

An empty **Build Results** tab indicates that there were no map build errors.

During map build, the logical interfaces within map rules are analyzed. Analysis includes checking for missing rules, invalid rules, invalid card definitions, verifying map references, and verifying the arguments of functions.

Rule Bar

The Rule Bar is one of the main components of the Map Designer user interface. The Rule Bar is used to display and enter map rules for the selected object on the output card in the To window.

By default, color coding is enabled in the Rule Bar. By default, the Rule Bar is docked beneath the toolbar. The Rule Bar can be resized or it can be undocked (floated) and moved. The Rule Bar is where you enter the map rules for each output data object.

When you select a rule cell in the To window, the rule appears in the Rule Bar. Changes to map rules are made and committed in the Rule Bar.

Non-printable characters are not displayed in the Rule Bar, but display in the rule cell on the To window as .

- To resize the Rule Bar, drag the lower border of the Rule Bar to the desired dimension.
- The default Commit key in the map rule is Enter. To enter a new line hold down the Ctrl key and press Enter. If Tab is the commit key, simply press Enter.
- To move the Rule Bar, first disable docking (right-click on the Rule Bar border and click **Allow Docking** to remove the checkmark.) You can then move the Rule Bar window anywhere on the desk top.
- To enter the object name of any data object, drag it from the map card into the Rule Bar.

You can type in extra spaces in a map rule to make it easier to read. Line feeds are not required, but may improve the way the rules are displayed. In addition, you can create a new line where you want one.

Text literals enclosed in quotation marks may not contain new lines.

Rule bar docking display option

The rule bar can be shown or hidden with the Rule Bar command on the **View** menu or from the rule bar's context menu (as shown below). The **Rule Bar** can be presented as a docked window or a floating window. Choose the display option from the rule bar's context menu.

To access the context menu when the rule bar is docked

Right-click on the rule bar's side border when docked.

To access the context menu when the rule bar is floating

Right-click beneath the title bar of the **Rule Bar** window.

The rule bar can be docked at the top of the Map Designer window (default), or undocked and moved around.

To dock or float the Rule Bar window

1. Right-click on the rule bar.
2. From the context menu, select **Allow Docking** .
If a check mark is displayed next to the **Allow Docking** command on the context menu, the rule bar can be docked.
3. After selecting **Allow Docking**, you can toggle between a docked rule bar and a floating window by double-clicking the top border of the rule bar.
4. Double-click the title bar of the rule bar so that the rule bar is again docked.

Showing and hiding the rule bar

The rule bar can be shown or hidden with the Rule Bar command on the **View** menu or from the rule bar's context menu.

To show or hide the Rule bar:

1. From the **View** menu, select **Rule Bar**.
A check mark appears next to **Rule Bar** in the **View** menu indicating if the **Rule Bar** is displayed.
2. To hide the rule bar when it is displayed, select **Hide** from the context menu of the rule bar. Or, from the **View** menu, select **Rule Bar** and clear the **Rule Bar** selection from the **View** menu.

Movable Windows

Windows within the Map Designer can be docked, or undocked and moved to other locations. You can move the Navigator, the **Rule Bar**, the properties window, any floated tab on the Organizer window, and toolbars. For example, you can float the **Rule Bar** and move it around.

To adjust the docking of a window that is docked:

Right-click on the docked window border and click **Allow Docking** to turn docking on or off from the context menu.

Map Designer toolbar

The toolbar is a part of the Map Designer window that provides you with quick access to various tools to invoke Map Designer actions while you work with map files and map cards. The left side of the toolbar provides general tools, such as create, open, and save. The remaining tools are specific to the Map Designer. The applicability and behavior of each tool depends on the kind of component selected and its current state or view.

The toolbar can appear in various positions within the application window.

See "Menu Commands and Tools" for a description of each tool.

Title bar

The title bar of the Map Designer window shows the full path of the active map source file, and the name of the selected map.

Status bar

The status bar displays contextual information such as descriptive messages about a selected menu command or tool or information about the current state of an operation. The message Ready indicates your may perform your next action.

The status bar can be hidden by disabling the Status Bar command on the **View** menu.

Menu commands and tools

Actions can be performed in the Map Designer using menu commands, tools, and shortcut keys. Not all menu commands have corresponding tools, and not all tools are represented in the command menus. When working with maps, there are several ways you can activate commands:

- Selecting functionality using the menu bar
- Right-clicking on any entry in the Navigator to display its context menu
- Right-clicking on the title bar of the input and output cards to display its context menu
- Right-clicking on any data object in the input or output cards to display its context menu
- Right-clicking in the rule bar in the output card to display its context menu
- Clicking the tools on the toolbar
- Double-clicking objects in the Navigator

Map Designer commands are available as listed above; however, most procedural information in the help uses the menu access as a default method. Please use the access method most convenient for you.

Map Designer menus

The Map Designer menu bar provides the common menu structure including the commands generally used with Windows programs as well as commands specific to the Map Designer.

File menu

The **File** menu provides the commands that are generally available in Windows applications.

Using the keyboard, press Alt F to view the **File** menu.

Command	Key stroke	Description
New	Ctrl+N	Creates a new map source file (.mms)
Open	Ctrl+O	Opens an existing map source file (.mms)
Close		Closes the selected map source file (.mms)
Save	Ctrl+S	Saves the selected map source file (.mms)
Save As		Saves the selected map source file under a different name
Source Control	Alt+F	
Create Project	Alt+L, C	Creates a new project
Open Project	Alt+L, O	Opens a project
Get Latest Version	Alt+L, G	Gets the latest version of the selected file

Command	Key stroke	Description
Check Out	Alt+L, E	Checks out the selected file
Check In	Alt+L, I	Checks in the selected file
Undo Check Out	Alt+L, U	Undoes the check out of the selected file
Add to Source Control	Alt+L, A	Adds selected file to source file
Remove from Source Control	Alt+L, R	Removes selected file from the source control
Show Workfiles	Alt+L, F	Shows all work files in the project
Show History	Alt+L, H	Shows the history for the selected files
Show Properties	Alt+L, P	Shows properties for the selected file
Refresh Status	Alt+L, S	Refreshes the status on all the files
Map Source File Differences	Alt+F, M	Shows the results of a comparison of selected map source files
Print	Ctrl+P	Displays a dialog box that allows you to specify the print options and print these options for the selected map(s)
Print Setup	Alt+F, R	Displays a dialog box that allows you to configure printer options
List of recently used files	Alt+F, <i>n</i>	Selecting a specific map source file from this list opens it
Exit	Alt+F, X	Quits the application and prompts to save changes

Edit menu

The **Edit** menu provides the editing commands generally available in Windows applications. It also contains the **Find** and **Replace** commands, which are very useful for modifying map attributes.

Using the keyboard, press Alt E to view the **Edit** menu.

Command	Key stroke	Description
Undo	Ctrl+Z	Reverses one or more multiple actions
Cut	Ctrl+X	Deletes the selection and places a copy on the clipboard
Copy	Ctrl+C	Copies the selection to the clipboard
Paste	Ctrl+V	Inserts the contents from the clipboard
Find	Ctrl+F	Displays the Find dialog box which allows you to locate specified information in the active window

Command	Key stroke	Description
Replace	Ctrl+H	Displays the Replace dialog box which allows you to locate and replace specified information in the active window.
More Find/ Replace	Alt+E, M	Displays the More Find/Replace dialog box for changing map attributes. The More Find/Replace operation searches an entire map source file.

View menu

The commands on the **View** menu enable you to control what is displayed in your Map Designer environment.

Using the keyboard, press Alt V to view the **View** menu.

Command	Key stroke	Description
Toolbar(s)	Alt+V, T	Displays the Customize dialog box for toolbars that allows you to select the toolbars to be displayed and to adjust the tools displayed on specific toolbars
Status Bar	Alt+V, S	Shows or hides the status bar that displays information is at the bottom of the Map Designer window
Rule Bar	Alt+V, R	Shows or hides the rule bar
Navigator	Alt+V, N	Shows or hides the Navigator

Map menu

The **Map** menu provides commands that initiate actions for a selected map.

Using the keyboard, press Alt M to view the **Map** menu.

Command	Key stroke	Description
New	Ctrl+M	Activates the Create New Map dialog box to provide a name for a new map
Reference	Ctrl+K	Opens the Map Reference Selection dialog from where you can create references to maps in other map source files.
Delete	Delete	Deletes the selected map(s)
Copy	Alt+M, Y	Copies the selected map to another map source file or within the same source file under a different name
Merge	Alt+M, M	Merges the selected map(s) to a different map source file

Command	Key stroke	Description
Rename	Alt+M, R	Displays the Rename Map dialog box to rename the selected map
Copy Data Audit Settings	Alt+M, A	Displays the Copy Data Audit Settings dialog box in which you can copy the data audit settings of the selected map to another map
Build	Ctrl+B	Builds the selected map(s)
Build for Specific Platform	Alt+M, S	Builds the selected map(s) for the specified platform
Run	Ctrl+R	Runs the selected map(s)
Run Results	Alt+M, T	Displays the Run Results dialog box in which you can view the run results of the selected map(s)
Close All Run Results	Alt+M, C	Closes all run results windows for the selected map
Organizer	Alt+M, O	Displays the Organizer window for the selected map
Where used	Alt+M, W	Shows where the selected map is used
Settings	Alt+M, G	Displays the Map Settings window for the selected map
Export	Alt+M, E	Exports the selected map to an XML file
Import	Alt+M, I	Imports the selected xml file into the Map Designer

Card menu

The **Card** menu provides commands that are performed on input and output cards.

Command	Key stroke	Description
New	Ctrl+E	Displays the Add Input Card or Add Output Card dialog box in which you can create a new card in the selected window
Edit	Alt+C, E	Edit the definition of the selected card
Delete	Delete	Delete the selected card

Command	Key stroke	Description
Convert to Input/ Output	Alt+C, V	Convert the selected input card to an output card or an output card to an input card
Copy	Alt+C, Y	Copy the selected card to another map or within the same map, under a different name
Reorder	Alt+C, R	Reorder the input and output cards
Open MDQ File	Alt+C, M	Opens the selected MDQ file in the Database Interface Designer

Rules menu

The **Rules** menu provides commands that are used for map rules.

Using the keyboard, press Alt R to view the **Rules** menu.

Command	Key stroke	Description
Insert Function	Insert	Inserts the function in the current map rule bar at the cursor position.
Insert Symbol	Alt+R, S	Displays the Symbols dialog box for inserting special characters into map rules.
Insert NONE if Empty	Alt+R, N	Automatically inserts the characters =NONE in all empty rule cells of the selected map.
Delete	Delete	Clears all text from the selected rule bar.
Properties	Alt+R, P	Displays the Rule Properties dialog box in which you can view the properties for objects, maps, and functions for the current rule.
Show All	Alt+R, A	Expands the output card display to show all the components that have map rules.
Goto Functional Map	Alt+R, G	Calls up all associated functional maps associated with the currently selected map source file.
Functional Map Wizard	Alt+R, W	Displays the Functional Map Wizard to automatically create functional map(s) based on the currently selected map rule.

Tools menu

The **Tools** menu contains commands that initiate actions to define shortcuts and display options.

Using the keyboard, press Alt L to view the **Tools** menu.

Command	Key stroke	Description
Short Cuts	Alt+L, S	Displays the Shortcut Keys dialog box in which you can assign shortcut keys or key combinations to specific Map Designer operations.
Options	Alt+L, O	Displays the Options dialog box in which you can configure Map Designer options for the Navigator, confirmation and other options.

Window menu

The **Window** menu contains the commands that provide control of open windows.

Using the keyboard, press Alt W to view the **Window** menu.

Command	Key	Description
Close All	Alt+W, L	Closes all open windows.
Cascade	Alt+W, C	Arranges all open windows so that they overlap in a descending pattern.
Tile Horizontally	Alt+W, H	Arranges all open windows as horizontal, non-overlapping tiles.
Tile Vertically	Alt+W, V	Arranges all open windows as vertical, non-overlapping tiles.
Arrange Icons	Alt+W, A	Arranges all minimized windows in an orderly fashion at the bottom of the Map Designer window.
To	Alt+W, 1	Makes the To window the active window.
From	Alt+W, 2	Makes the From window the active window.

Help menu

The **Help** menu offers choices that display information about the Map Designer.

Using the keyboard, press Alt H to view the **Help** menu.

Command	Key stroke	Description
Contents	Alt+H, C	Displays the contents of the Help system which also lists the Help topics.
About Map Designer	Alt+H, A	Displays application-specific information.

Chapter 3. Map Designer basics

Map Designer file name extensions

The following is a list of the file name extensions used in the Map Designer:

Extension	File type description
.bak	Map source backup
.fnl	Master work file
.lxx	Input work
.log	Audit log
.mopt	Map source options file
.mmc	Compiled map
.mme	Map build analysis results
.mms	Map source
.mtr	Trace
.omm	Backup of a 1.4.x map source
.Oxx	Output work
.tmp	Temporary map work files

Note: Zero-byte temporary map work files might not be deleted if the Map Designer application is not gracefully disconnected.

Starting the Map Designer

The installation program adds an entry for the Map Designer to the program folder. For a default installation, from the Windows **Programs** menu, navigate to WebSphere Transformation Extender **Design Studio** → Map Designer. The Startup window opens and the following options are available:

- **Open an existing map source file**
Browse for the map source file (.mms).
- **Create a new map source file**
Create a new map source file (.mms).
- **Open a recently used map source file**
Select one or more map source files (.mms) from the displayed file list. You can also double-click on a file from this list to open it.

The Startup window can be disabled by selecting **Do not show this at startup**; however, you can always access this dialog from the **Help** menu by choosing **Startup Window**.

Configuring the Map Designer environment

Much of the Map Designer environment can be configured to accommodate your preferences in your working environment. For example, you can specify various user interface options, such as:

- Font type or color.
- How lines, windows, or toolbars appear.
- When confirmation messages are displayed.

You can also assign shortcuts for frequently used keystrokes.

To access the Map Designer configuration options:

From the menu bar, select **Tools** → **Options**.

The Options dialog box is displayed. The left pane of this dialog lists the categorized areas of the Map Designer for which you can customize settings.

General options

Use the General options in the Map Designer to specify values concerning the backing-up and saving of your map files, map rule compare preferences, and the font for the Select Type window.

General options:

- **Auto-save files every n minutes** (where n represents a number)
This number n is the time interval (in minutes) for automatically saving open map files.
- **Override maps on merge**
Enable this check box to specify whether you want to overwrite maps in the destination map source file when merging maps. The Override maps on merge check box is disabled by default.
- **Backup on save**
Enable this check box to specify whether you want a backup copy of each map source file to be created when the file is saved. The Backup on save check box is enabled by default.
- **Suppress build warnings**
Enable this check box to specify whether to suppress the checking of warning conditions and generation of build warnings during map compilation. The **Suppress build warnings** check box is disabled by default.
- **Ignore white spaces and tabs**
Enable this check box to specify whether to ignore white spaces and tabs in map rules when using the Map Source File Differences command.
- **Font for the Type dialog**
Select the font and font size to be used for text in the Select Type dialog box. Types are selected as part of card definitions.

From window options

For the **From Window** options, you can specify how to display objects in the From window.

From window options:

- **Select Font** drop-down list
The font selections apply to the font displayed in the card title or the font for the contents of the From window. Select **Card Title** or **Active Card**.
- **Font** button
Click this button to select the font and font size to be used for text in the **From** window.
- **Active card Color**
Click the down arrow to display the **Color** palette to change the color of the active card in the From window.
- **Max. no. of cards viewed**
Enter the maximum number of cards to display in the From window.
- **Show floating component type**
Enable or disable this check box to control the display of floating component types. The **Show floating component type** check box is disabled by default.

To window options

Use the To window options to specify how to display objects in the To window.

The fields in this dialog box are as follows:

- **Select Font** drop-down list
The font selections apply to the font displayed in the card title or the font for the contents of the To window. Select **Card Title** or **Active Card**.
- **Font** button
Click this button to select the font and font size to be used for text in the **To** window.
- **Active card color**
Click the down arrow to display the **Color** palette to change the color of the active card in the To window.
- **Max. no. of cards viewed**
Enter the maximum number of cards to display in the To window.
- **Show all rules**
Enable or disable this check box to expanded or collapsed display of map rules in the To window. When enabled, rule cells that contain map rules are automatically expanded when a map is displayed. When disabled, rule cells that contain map rules are collapsed when a map is displayed. The **Show all rules** check box is enabled by default.

Rule Bar options

Use the Rule Bar options to specify how to display objects in the rule bar.

General tab

- **Font** button
Click this button to select the font and font size to be used for text in the rule bar.
- **Commit rules with** drop-down list
Select to use the **Enter** (the default value), **Tab**, or both (**Enter** and **Tab**) key to commit a map rule.

- **Use ellipses**
Enable this check box to use ellipses instead of long path names when dragging or copying objects.

Color coding tab

- **Use color coding** check box
Enable or disable this check box to specify the use of color-coded display of elements in map rules.
- **Show errors with underline** check box
Enable or disable this check box to specify the use of underline on invalid portions of map rules.
- **Color Specifications** drop-down list
 - Select the element of the map rule for which you want to specify a color, and then click the down arrow to display the **Color** palette to change the color of the selected map rule element.

Navigator

In the Navigator options dialog box, you can specify how to display objects in the Navigator window.

Navigator options:

- **Font** button
Select the font and font size to be used for text in the rule bar.
- **Lines** group box
Select any of these three option buttons (**None**, **Solid**, and **Dotted** [the default value]) to determine the appearance of the lines in the Navigator window.
- **Show tool tips** check box
Enable this check box to display the name of each object when the cursor is held over it and the Navigator window is sized too small for the name to be completely displayed. The **Show tool tips** check box is enabled by default.

Confirmations

In the **Confirmations** options dialog box, you can select actions for which you want a confirmation dialog displayed before completing the action. The confirmation dialogs are available for map operations and card operations. All options are enabled by default.

Organizer window options

Use the Organizer window options to specify how to display objects in the Organizer window.

Organizer window options:

- **Tab** drop-down list
Select which tab of the Organizer window on which you want to change the **Font**.
- **Font** button
Click this button to select the font and font size to be used for text in the tab currently shown in the **Tab** field.

- **Show only on errors** check box

After a map is built, the **Build Results** tab on the Organizer window can be displayed always or only when a build error has occurred. Enable this check box to display the **Build Results** tab only when an error has occurred.

Run results

Specify how to display objects in the Run Results window.

The fields in this dialog box are as follows:

- **Font** button

Click this button to select the font and font size to be used for text in the Organizer window.

- **Hex Display** → **Show offset in decimal values** check box

The **Show offset in decimal values** check box allows you to specify whether (enabled) to display the data offset using decimal values in hexadecimal display.

Run options

Specify default map settings, display settings, and the default resource configuration file.

DisplaySettings

The **DisplaySettings** apply to any map that is run from the Map Designer. The **DisplaySettings** are used to configure how often to refresh the Command Server window when a map is running and when to close the window when a map completes.

RefreshRate

The **RefreshRate** specifies how often to refresh the Command Server window during map execution. The default **RefreshRate** is two times per second.

Value Description

n This is the number of times per second that a refresh of the Command Server window is performed during map execution.

AutoClose

The **AutoClose** options control when the Command Server window is closed after map execution.

Value Description

Never The Command Server window never closes and remains open after the map execution is complete.

Always

The Command Server window closes automatically after the map execution is complete.

Success

The Command Server window closes only if a map ran successfully.

Configuration file

Browse your file structure to select a default resource resolution file. If this value is not specified, the default Resource Configuration (resource.mrc) file specified in the Execution Section of dstx.ini file is used. A resource configuration file contains specifications for an engine such as the active virtual server(s) and its associated .mrn file.

Properties window options

You can specify the font and font size to be used for text displayed in the properties window.

Referenced Maps option

The **Referenced Maps** option allows you to choose a resource alias file for referenced maps.

Profiler options

Before profiling a map, you must enable a **Function Times** option or a **Type Times** option to see statistics in the output (otherwise the report is empty).

Function times

Enable the **All** or **Above** option to view function time statistics in the profiler report.

Option	Description
None	(Default setting) When None is enabled, function time statistics are not reported.
All	When you select All, each function and each function call is provided in the profiler output. Every occurrence of every function is displayed with the depth (level), number of iterations, elapsed time, map name, and type name.
Above	Use the Above option to obtain a smaller output file. When you specify a value in 1/10 milliseconds (10000 = 1 second), function times are only recorded when they exceed that value.
Summary	When you choose the Summary option, the report contains a summary for each function instead of listing every occurrence. A function is listed once with the total number of iterations and total elapsed time for all iterations. Sample report.

Type times

Enable the **All** or **Above** option to view type time statistics in the profiler report.

Option	Description
None	(Default setting) When None is enabled, type time statistics are not reported.
All	When you select All, each type per rule is provided in the profiler output. For each rule, the time, map name, and type name are reported.

Option	Description
Above	Use the Above option to obtain a smaller output file. When you specify a value in 1/10 milliseconds (10000 = 1 second), types times are only recorded when they exceed that value.
Summary	When you choose the Summary option, the report contains a summary of types per rule. A type name is displayed once with the total number of iterations and total elapsed time for those iterations. Sample report

File output

Option	Description
Fixed Width	This option produces a readable, flat file output.
Comma Delimited	This option produces a comma-delimited output which is useful for exporting to spreadsheet programs.

Debug options

The following options pertain to the Map Debugger.

- **Data Length:** Maximum size of the data (in bytes) displayed in the **Data** column of the Debug Rule window.
- **Breakpoint Color:** When a breakpoint is set on an output object, the object rule in the output card is highlighted in this color.

Search facilities in the Map Designer

The Map Designer supports standard search facilities. There are two Find commands: **Find** searches the selected object and **More Find/Replace** searches an entire map source file.

The Find command searches on:

- Object names in a card
- Map rules in a card
- A page in the organizer window

The More Find/Replace command searches on:

- All card names within a map source file
- All card type names within a map source file
- All output objects within a map source file
- All map rules within a map source file
- All type tree names within a map source file

Using the find operation

Use a Find operation to find text within object names in a card, within map rules in a card, or within a page in the organizer window.

To find entities using a text string search:

1. Select the area where you want to do the search.
For example, select the card to be searched.

2. From the **Edit** menu, choose **Find**.
3. In the **Find what** field, enter the text you want to find.
4. Optionally, enable the **Match whole word only** and/or the **Match case** check boxes.
5. In the **Direction** group box, enable either the **Up** or **Down** option button to specify the direction to search. **Up** (to search to the beginning of the file) or **Down** (to search to the end of the file).
6. Click **Find Next**.
7. Continue to click **Find Next** for as long as you want to search, then click **Cancel**.

If a match is located, the match is selected. Otherwise, there may be a message asking whether you want to continue searching.

Using a replace operation

The Replace operation replaces the found text in the **Unresolved Rules** tab in the organizer window, and the map rule in the rule bar.

To replace text:

1. Select the window where you want to do the search.
For example, the **Unresolved Rules** tab in the organizer window.
2. From the **Edit** menu, choose **Replace**.
3. In the **Find what** field, enter the text you want to find.
4. In the **Replace with** field, enter the text with which you want to replace it.
5. Optionally, enable the **Match whole word only** and/or the **Match case** check boxes.
6. Click **Find Next**.
7. Click **Replace** if you want to replace that occurrence and continue until you are finished replacing. Or, click **Replace All** to replace every occurrence.

Using a more find/replace operation

The More Find/Replace operation operates on the entire contents of a map source file. For example, you can search through cards and replace filenames ending with .txt with .dat. You could replace all card types named **Record Set Data** with **Record Data**. You could find card names that include the word **Test**.

To find text within an entire map source file

1. Select the map you want to search.
2. From the **Edit** menu, choose **More Find/Replace**.
The More Find/Replace dialog box is displayed.
3. In the **Find** field, enter the text you want to find.
4. From the **Find within** list, choose where you want to find the text - within card names, card type names, output names, map rules, or type tree names.
5. Optionally, enable the **Match whole word only** and/or the **Match case** check boxes.
6. Click **Find All**.

To replace text within an entire map source file

1. In the **Replace with** field, enter the text you want to replace.
2. In the **With** field, enter the text with which to replace it.
3. Disable the check box for the cards in which you do not want to replace text.
4. Click **Replace**.
5. Click **Close**.

To go to a card listed in the More Find/Replace dialog box, press **Alt** and double-click the card name in the More Find/Replace dialog box.

Areas to search

The areas the More Find/Replace command can search are listed below.

Find within:

Searches:

Card Name

Card names

Card Type Name

Card type names

Output

Output objects that have map rules

Rule

Map rules

Type Tree Name

Type tree names

It is not possible to replace text when **Output** is chosen in the **Find within** field.

Printing

For a map selected in the Navigator, you can print:

- Map settings
- All of the details of a map and the print options enabled in the Print dialog box.

With current focus on a card, you can print:

- A compositional view of data objects on the card

With current focus on the properties window, you can print:

- Type properties from the Map Designer while viewing them in the properties window.

Printed map details

The following items can be printed for a map in the Map Designer:

- Map name
- Map-specific options
 - Unresolved rules
 - Remarks
 - Audit log
 - Audit settings
 - Trace file output

- Build results
- Input cards (**CardName**, **TypeTree**, and **Type**)
 - Detail of data objects on the card
- Output cards (**CardName**, **TypeTree**, and **Type**)
 - Detail of data objects on the card and related map rules
- Compositional display of data objects on the card

Printing map details

You can print out map details from the Map Designer.

To print map details:

1. Select a map in the Navigator.
2. Right-click the map and choose **Print** from the context menu.
The Print dialog box is displayed.
3. Enable the check box for the desired print options.
4. Click the **Font** button to change the font if desired.
5. Click **OK**.

Printing map settings

The Map Settings dialog box can be printed as displayed. The Expand All command should be used on all map settings to print all map settings.

To print map settings:

1. Select a map in the Navigator.
2. Open the map settings.
3. For the map settings to be printed, right-click the desired top-level map settings and choose **Expand All**.
4. Right-click any map setting and choose **Print Properties** from the context menu.

You can also export a map to obtain information about map settings.

Printing input and output card information

The input and output cards can be printed as displayed. The Show All command should be used to print all data objects on the card.

To print an input or output card:

1. Place current focus on the input or output card.
2. Right-click and choose **Show All** to show all nested data objects.
3. Right-click the card title bar and choose **Print**.

You can also export a map to obtain card settings documentation.

Printing Organizer information

The **Organizer** window has six tabs. You can print out information about each tab.

To print a tab in the organizer window:

1. Display the desired tab in the **Organizer** window.
2. Right-click and choose **Print** from the context menu.

The current tab on the **Organizer** window is printed.

The print options are not configurable at print time. Use the **Tools Options** → **Organizer** font option to control the font of the printed organizer tabs.

Printing run results

After a map is run, you can print the run results for each output card.

To print the currently active run results window:

1. Select **Map** → **Run Results**.

The Run Results dialog box is displayed.

2. Enable one or more of the check boxes next to each output card.
3. Click **OK**.

The results are displayed.

4. Right-click the desired run results window and choose **Print** from the context menu.

The run results are printed.

Chapter 4. Maps and map source files

The tasks involved in creating and managing map files include:

- Creating map source files (.mms)
- Creating maps in the map source file
- Creating input and output cards for the map
- Building compiled maps (.mmc)
- Running maps
- Viewing run results

A map defines how to generate a data object of a certain type or multiple independent data objects, each of a certain type. A map contains input and output cards that transform data content from source formats to destination formats according to rules contained in the map.

A map source file is used to store maps that are related to one another. A map source file is a file you create and save using the commands on the **File** menu in the Map Designer. A map source file has the file name extension **.mms**. A map source file maintains maps in source format. Map source files contain maps and functional maps.

After starting the Map Designer, the Navigator lists one or more map source files depending on whether you selected to create a new map source file or to open one or more existing map source files. The full path and filename of the map selected in the Navigator appear in the title bar of the Map Designer window.

After a map source file is created, the map source file name and icon appear in the List view of the Navigator.

The Composition view of the Navigator displays a map only after an input or output card has been added to the map.

Maps are not executed directly from a map source file. You must first build (compile) a map before you can execute it. Compiled maps have the file name extension **.mmc**.

Creating a map source file

Map source files contain one or more maps. It is logical to organize your maps in map source files according to use or purpose.

To create a map source file:

1. From the **File** menu, choose **New**.
The Save As dialog box is displayed.
2. Specify the location of the map source file by browsing your file structure.
3. In the **File name** field, enter the name of the map source file.
4. Click **Save**.

The map source file displays in the Navigator.

Creating a map

One or more maps are contained in map source files. The desired map source file must be selected in the Navigator before you can create a map.

To create a map:

1. Select the desired map source file in the Navigator.
2. From the **Map** menu, choose **New**.
The Create New Map dialog box is displayed.
3. Enter the name of the map to be created in the **New map name** field.
4. Click **OK**.
The map appears in the Navigator.

Map name guidelines

The guidelines to follow when naming a map are:

- A map name can be up to 32 characters in length.
- A map name cannot be a reserved word.
- A map name cannot contain only digits and/or periods.
- A map name can contain combinations of the following types of characters:
 - Letters
 - Digits
 - ASCII characters 128-255
 - These special characters: ~ # % \ ? _ ' . ; `
 - Double-byte characters (Japan edition only)

It is helpful to use an action-oriented map name that describes the purpose of the map. For example: **CreatePO**.

Map source file differences

Map source files contain maps, and maps contain cards. Map source file differences are compared for each map source file at the map and card levels.

Map source files are considered "different" when:

- A map exists in one map source file and does not exist in the other map source file.
- Any of the maps are different.

Maps are considered different if any of the following differences exist:

- Any card is different
- Any card exists in the first map and does not exist in the second map
- Any audit setting is different
- Any remark is different
- Any unresolved rule is different
- The order of the cards in the map is different

Cards are considered different if:

- The card settings are different
- The map rules on output cards are different

In addition to the map source file differences, the source file differences feature identifies differences for type trees, system definition files, and database/query files. You can view the results of the map source file differences.

Compare map source files in the Map Designer.

Colors indicate the status of the differences.

- **Differences** are red.
- **Additions** are blue.

Comparing map source files

Map source file differences are compared for the selected map source files.

To compare map source files:

1. From the **File** menu, choose **Map Source File Differences**.
The Select First File dialog box is displayed.
2. Browse your file system and select the first map source file to be compared.
3. Click **Open**.
The Select Second File dialog box is displayed.
4. Browse your file system and select the second map source file to be compared.
5. Click **Open**.
The progress of the comparison is shown briefly in the Map Source File Differences Analysis dialog box and then the Map Source File Differences window is displayed.
Resize and position window as desired to view all differences.

Maps and cards within maps with different names can be compared.

Comparing objects with different names

After the Map Source File Differences window is displayed, the settings will show only for the first file selected.

To view the settings of objects with different names:

Press the Alt key while clicking any object in the right pane of the Map Source File Differences window.

The settings for that object are displayed.

Viewing map source file differences

After two map source files are compared, the Map Source File Differences window is displayed. Resize and position window as desired.

The Map Source File Differences window has four panes. The contents of the two map source files being compared appear in the top portion of the window and are displayed alphabetically from top to bottom. This view presents a detailed list of the maps contained in the map source file, and the input and output cards for each map.

Colors provide an indication of differences in map source files.

- **Differences** are red.
- **Additions** are blue.

To view map source file differences:

1. Place current focus in the Map Source File Differences window.
2. Select any map or card to view the differences.

The settings display in the lower windowpanes. Settings that have differences are indicated in red.

3. Press F8 to view the next difference.

Press F7 to view the previous difference.

In the Map Source File Differences window, you can view map and card settings on any element for the compared map source file, even when differences do not exist. Press Alt while clicking the object to view the settings of the different object.

Troubleshooting tip

If map differences are indicated but not viewable, there is a possibility that if the map was exported, the adapter specified in the map source file was no longer supported or not installed at the time of export. When the map source is imported and then compared, a difference is detected but is not viewable.

Resizing the map source file differences window

File differences are displayed in a split-screen window that is divided into four sections. You can resize any section of the window as needed.

Map differences

- **Map settings**

If any map settings of the compared map files are different, the **name** of the map with differences appears **red** in the Map Source File Differences window.

The **MapSettings** tab in the lower windowpane is red.

- **Unresolved map rules**

If any unresolved map rules of the compared map files are different, the **name** of the map with differences appears **red** in the Map Source File Differences window.

The **Unresolved Rules** tab in the lower windowpane is red.

- **Remarks**

If any remarks in the Organizer window of the compared map files are different, the **name** of the map with differences appears **red** in the Map Source File Differences window.

The **Remarks** tab in the lower windowpane is red.

- **Data Audit Settings**

If any data audit settings defined in the Organizer window of the compared map files are different, the **name** of the map with differences appears **red** in the Map Source File Differences window.

The **Data Audit Settings** tab in the lower windowpane is red.

- **Card Settings**

If any card settings of the compared map files are different, the **name** of the card with differences appears **red** in the Map Source File Differences window.

The **Card Settings** tab in the lower windowpane is red.

Chapter 5. Input and output cards

A map defines how to generate data objects of a certain type. A map contains input and output cards that transform data content from source formats to target formats. This data transformation occurs according to mapping rules on the data objects in the output card in the map.

Input and output cards represent data objects. Each card represents one data object, which is defined as a particular type. There are two types of map cards: input cards and output cards.

- An input card contains the complete definition of an input for the map including information such as source identification, retrieval specifics, and the behavior that should occur during processing.
- An output card contains the complete definition of an output for the map including information such as target identification, destination specifics, and the behavior that should occur during processing.

Each input of data and each output of data requires content definition settings. The content of each card is specified as a type in a type tree.

Card overview

Each map may contain none or more input cards and one or more output cards. The maximum number of input cards in a map is 100. The maximum number of output cards in a map is 100.

Each card represents one data object, identified by selecting a type from a type tree. A card may represent an object in a row, a row in a record, the entire record, or the entire file.

Each card specifies a type tree and a type as part of the card definition. This type, and the type tree that contains it, is the link between the Type Designer and the Map Designer that defines the structure and properties of the data in the card.

- The input card represents input data. Input data validation occurs in the Type Designer.
- The output card represents output data and contains the map rules that transform that data. Therefore, only the output card has a rule column.

Card specifics

- More than one card in a map can reference the same type but the cards must have different card names.
- Each card is assigned a number when it is created. Cards are numbered sequentially, beginning at 1. The card numbers determine the order in which the cards and the data in those cards is evaluated. The card number is important when a map is used as a functional map. To reorder the cards, use the Reorder command on the **Card** menu.
- The structure of the card object appears in the card. An icon represents each component that the entire card data object is composed of. Each icon in the input and output card represents a type that has been defined in the Type Designer.

- The output column in the output card shows the structure and the names of the components of the output data object.
- In the case of partitioned groups, the structure shown is the partitioned group and its subtypes. An icon is shown for each component and each partition that compose the entire card object.
- Each output name appears in a separate cell in the output card.
- The rule column contains a rule cell for each output object. Enter map rules to define how to generate the data object of this card.
- There are several ways to view input and output cards.

Viewing input and output cards

Maps contain one or more cards. To display a card:

- Click the card in the To or From window.
- Scroll through the cards using the arrows at the lower right corner of the window.
- Double-click a card in the List view in the Navigator.

Compositional hierarchy

The compositional view of the cards shows the contents of the type definitions. Types are arranged in a compositional hierarchy, which shows the structure, layout or composition of the data. This hierarchy is different from the hierarchy in a type tree, which is a classification hierarchy.

The data is defined in the Type Designer according to specific properties. The types are arranged on the input and output cards in the order they appear in the data. This compositional hierarchy shows the structure of the data.

For example, the **ContactFile** input card #1 shown in the following map source file represents the input data of the map. The **Contact(s)** group type contains the item types that together make up the contact record. The **Label** output card represents the output data of the map. The **Label** group type contains the item types that together make up the output data.

Creating input and output cards

The create card function relates directly to which window is active when the new card is created. To create an input card, the From window must be active. To create an output card, the To window must be active.

The card is created in the currently selected map. The full path and name of the currently select map appears in the title bar of the Map Designer.

To create an input card

1. To create an input card, select the From window.
2. From the **Card** menu, choose **New**.
The Add Input Card dialog box is displayed.
3. Define the card by entering the appropriate information for each setting.
4. Click **OK**.

To create an output card

1. To create an output card, select the To window.
2. From the **Card** menu, choose **New**.
The Add Output Card dialog box is displayed.
3. Define the card by entering the appropriate information for each setting.
4. Click **OK**.

Editing an input or output card

Any card setting may be changed: the card name, the card type, or the card source or target information.

To edit a card:

1. Select the card you want to edit.
2. From the **Card** menu, choose **Edit**.
The **Edit Input Card** or Edit Output Card dialog box is displayed.
3. Make necessary changes to the card setting values.
4. Click **OK**.

Two quick access methods to the Edit Card dialog box:

- Double-click the title bar of the card.
- Right-click the title bar of the card and select **Edit** from the context menu.

Chapter 6. Card settings

You configure how to run a map by specifying map settings in the Map Settings dialog box, defining **SourceRule** card settings on the input card, and defining **TargetRule** card settings on the output card. All of the card settings are on the **Add (or Edit) Input Card** and **Add (or Edit) Output Card** dialog boxes.

Each input of data and each output of data require content definition settings. Card settings define the data object the card represents, and how the data is retrieved or routed.

Card settings can be exported. See "Exporting a Map Source File" for details about exporting map card settings.

Schema

The **Schema** for input and output cards defines the **CardName**, the **TypeTree**, and the **Type**.

CardName

Each card in a map has a unique name to distinguish cards in the map. Precise naming of the card is practical and useful. A good practice is to name the card the same name as the data object it represents. Follow the card name guidelines.

Although cards can be named most anything you want, typically you would want to use a name that *describes the data*, such as **InputFile**, **ClaimData**, **ItemMasterRow**, and **CreditApprovalMsg**.

To specify the CardName:

1. For the **CardName** setting, enter the desired card name.

The name of the card displays next to the card number in the card title bar and as the top-level type icon in the card.

CardName guidelines

Card names follow these guidelines:

- A card name can be up to 32 characters in length.
- A card name cannot be a reserved word.
- A card name cannot contain only digits and or periods.
- A card name can contain *only* the following:
 - Letters
 - Digits
 - ASCII characters 128-255
 - These special characters: ~ # % \ ? _ ' . ; `

TypeTree

The **TypeTree** setting is the name of the type tree file (**.mtt**) that contains the type describing the data represented by the card.

Cards represent a particular data object. Therefore, each input card and each output card is associated with a type in a type tree that describes the content of that data. Therefore, each card includes a type tree and type name as part of its definition. This type tree is the *link* between the Type Designer and the Map Designer that communicates the structure and properties of the data to the Map Designer.

TypeName

The **TypeName** setting specifies the name of the type within the type tree, which describes the data represented by the card. This type must exist in the type tree specified with the **TypeTree** setting.

To select the **TypeName** for the card:

1. For the **TypeName** setting, click the browse button to view the Select type dialog box.
2. Select the desired type or use the **Find** button to search the type tree for a particular type.
3. Click **OK**.

The name of the selected type appears.

SourceRule input card settings

SourceRule settings specify what data to retrieve, where to get it, how much to get, and what to do when an error occurs.

The **SourceRule** settings define how the input is retrieved. The data object specified in the input card defines what data is retrieved. The **SourceRule** settings define how the data is retrieved.

To define card-specific data retrieval behavior, specify the **SourceRule** settings for each input card. An input card represents an input source. The content of an input is a *data object* such as a table of patient records in a relational database, a message that contains a mortgage application, or a file of item master update records that is transferred from another location using the FTP adapter.

The content of an input card is defined with the **CardName**, **TypeTree** and **Type** settings. These settings define what the data is, but does not specify how to retrieve the data. The **Source**, **FetchAs**, **WorkArea** and **FetchUnit** card settings are some of the **SourceRule** settings that define how the data is retrieved for each input card.

To define **SourceRule** settings:

1. Select the input card.
2. From the **Card** menu, choose **Edit**.
The Edit Input Card dialog box is displayed.
3. Enter desired values for **SourceRule** settings.

SourceRule Setting overrides

Input card settings are stored in the compiled map. You can override the setting for each option from other applications, including the Command Server, Launcher, and Integration Flow Designer.

Execution commands can be used to override the map settings or card settings compiled into the compiled map when a map is run. See the Execution Commands documentation for a complete description of the override execution commands.

The **SourceRule** card settings and their override execution commands are listed in the table below:

SourceRule Card Setting	Override Execution Command
FetchAs	Not available
Backup	Not available
WorkArea	Available (-W)
Source	Available (-IA), (-ID), (-IE), (-IF), (-IM)
FilePath	Available (-IA), (-ID), (-IE), (-IF), (-IM)
OnSuccess	Available (-I...X)
OnFailure	Available (-I...B)
Retry	Available (-I...XRcount:interval)
Warnings	Not available
Scope	Not available
FetchUnit	Not available

FetchAs

The **FetchAs** setting on an input card specifies how the data will be retrieved when the map is executed.

Value	Description
--------------	--------------------

Integral	Data is retrieved once (in its totality) from the data source. If another input is set to Burst mode, the same integral data for this card is used for each burst. When an input FetchAs is integral, the Source is requested to retrieve data once for the entire mapping.
Burst	Data is retrieved in incremental units from the data source. The unit of data retrieved for each burst is adapter-specific, and is specified as the Source FetchUnit . When an input FetchAs is Burst , the data is retrieved once per burst.

When an input **FetchAs** is **Integral**, any failure action causes the entire map to fail.

- When **FetchAs** is **Integral**, the maximum size of a file that a map can process is 2 GB.

- When **FetchAs** is **Burst**, the maximum size of a file that a map can process is much larger, but a maximum of 2 GB per burst.

FetchAs > WorkArea

Each time a map is executed, information about the data and map is kept in a workspace and is used as the input data is validated and the output data is built. This workspace information can either be stored in files or written to memory.

A card's **WorkArea** setting specifies whether to reuse that work area on subsequent executions of the same map. If you are repeatedly going to run the *same* map using the *same* input data more than once, setting the **WorkArea** card setting value to **Reuse** increases the speed of map execution because data validation for this input is not repeated. There are guidelines for reusing work areas.

Value Description

Reuse Reuse the work area. After a map runs for the first time, the work area that is created for the input card is not deleted. Then, on subsequent executions of the same map, the data for this input card is *not* validated, and the work area information for the card is retrieved from the existing work area.

!Reuse

Do not reuse the work area. The input card's **WorkArea** is created for each map execution.

Guidelines for reusing work areas

Use the following guidelines when defining whether to reuse workareas. The **WorkArea** card setting depends upon the map setting for **WorkSpace Location** and **PageSize** and **PageCount** settings.

- **WorkSpace Location** setting. If the map setting for **WorkSpace** is **File**, the input card's work area can be reused. If the map setting for **WorkSpace** is **Memory**, the input card's work area can be reused only when the map is initiated using the **RUN** function. It cannot be reused from an API.
- **WorkSpace PageSize** and **PageCount** settings. When reusing work files, the paging configuration (page size and number of pages) for the first execution and subsequent executions of the map *must* be the same.
For example, if the first execution (when the work files are created) uses a paging configuration of eight pages of 64K each, the subsequent executions (where the work files are reused) must use the same paging settings, in this case eight pages of 64K each.
- **For One Map Only**. The **WorkArea** card setting applies to a given map only. For example, it does *not* apply to two different maps that using the same input.
- **FetchAs = Integral**. When a map is executed and the input card **FetchAs** = **Integral**, the work area is automatically reused for each burst. If **WorkArea** = **Reuse**, that work area is saved on completion of the map execution.
- **FetchAs = Burst**. When the input card has a **FetchAs** of **Burst**, a new work area is created for each burst. If **WorkArea** = **Reuse**, the last work area is saved on completion of the map execution.

FetchAs > FetchUnit

The **FetchUnit** setting only applies to input cards. The **FetchUnit** setting defines the number of units of data to retrieve each time a request for data is made to the

adapter. **FetchUnit** is primarily used when the **FetchAs** setting is set to **Burst** although it may also be used in other situations. The default value for **FetchUnit** is **S** (unspecified all).

A specific **FetchUnit** value allows you to aggregate, or "batch", a set of physical units (such as messages on a queue) into a logical unit of work. The unit of data is adapter-specific.

Value Description

S Get all objects, database rows, messages on a messaging queue, or files.

1 - nmmn

Positive integer representing the logical or physical unit of data.

For messaging adapters, **FetchUnit** defines the number of messages to retrieve.

For **File** or **Echo**, **FetchUnit** defines the logical unit of data.

For database adapters, **FetchUnit** defines the number of rows.

FetchUnit details

The unit of data is adapter-specific.

- If the adapter is **File**, the **FetchUnit** refers to a logical unit of data and divides map execution into valid units of work based on a logical unit of data.

For other adapters, the **FetchUnit** refers to a physical unit of data.

- For messaging adapters, the **FetchUnit** setting determines the number of messages per burst.
- For database adapters, the **FetchUnit** setting determines the number of database rows.

When an input **FetchAs = Integral**, the **Source** is requested to retrieve data once for the entire mapping.

When an input **FetchAs = Burst**, the data is retrieved once per burst.

For example, a database has the rows sorted in reverse chronological order by date. You want to map only the first row because that row contains the most recent data. Set the input **FetchAs = Integral** and **FetchUnit = 1**. Only the first row in the database is accessed.

Using different **FetchUnit** values for the same data object on multiple input cards has limitations.

FetchUnit with multiple input cards

In a case where more than one input card in a map references the same data object, the values of the **FetchAs** and **FetchUnit** card settings must be the same on all cards that reference that data object. Only one instance of the type tree component structure is stored with the compiled map, therefore the card settings that control that component structure must be the same on all cards using that data object. The type tree component structure stored with the map has one number that represents the maximum instances of that component. The **FetchUnit** setting changes this number.

For example, a map contains two input cards that both reference the **ContactFile** data object. This data object may be the card object (as specified with the **TypeTree** (Contact.MTT) and **Type** (**ContactFile Data** settings, as shown below) or may be a component of the card object.

To achieve desired data retrieval results, you must set the **FetchAs** and **FetchUnit** values the same on both cards that use the same type (**ContactFile**).

If different **FetchAs** and **FetchUnit** values are specified for the same data object on two different input cards in the same map, the **FetchAs** and **FetchUnit** values specified in the first card are used.

GET

The **GET** settings define all of the input data settings.

Source

The **Source** card setting identifies the source of the input data. The data may be in a file, from a specific application, in a database, from a message queue, or from other possible sources. Defining the adapter source specifies where to go to get the data and specifics on which data to get.

The meaning of the various adapter source settings is specific to the adapter being used to get the input data.

The maximum size of a file that a map can process in integral mode is 2 GB. In burst mode, the maximum size is much larger, but can only be 2 GB in one burst.

Source > FilePath

When **Source = File**, the **FilePath** setting specifies the name and path of the input data file. Enter the filename or browse your file system to select the input file.

Source > Command

When **Source = various resource adapters**, the **Command** card setting provides the adapter-specific commands to connect to the data source and retrieve the input data. For example, if the input data is coming from a file, the **Command** specifies the name of the input file. If the input data is coming from a database, the **Command** might include the database name, user ID, password, and so on.

The command syntax for the source and target adapter commands is detailed in the adapter-specific documentation. The maximum length of a source or target adapter command is 260 bytes. Limitations to the maximum adapter command length may apply to some adapters.

DatabaseQueryFile

When **Source = Database**, the **DatabaseQueryFile** settings are used to specify the database specific settings: the name of the desired database/query file, the database name, and the query.

DatabaseQueryFile > Database

Specify the database that contains the desired query definition.

DatabaseQueryFile > Query

Specify the name of the desired query.

DatabaseQueryFile > File

Specify the database/query file (.mdq) that contains the desired source database and query definition.

Transaction

The **Transaction** settings include the input data **OnSuccess**, **OnFailure**, and **Scope** settings.

OnSuccess

The **OnSuccess** setting specifies the location of the source data when a map, a burst, or a card successfully completes. The availability of the **OnSuccess** setting is adapter-specific.

The **OnSuccess** setting works with the **Scope** setting, which defines when to apply the action specified with the **OnSuccess** setting.

Value Description

Keep The input data should *not* be removed from its source.

For example, if the input is a messaging adapter, and the map successfully completes, do *not* remove the messages from the message queue.

KeepOnContent

The input data should not be removed unless it has no content.

For example, if the input is a file, and the map completes successfully, but the input file had no data (it was empty), delete the input file. Otherwise, keep the input file.

Delete Delete the input data or remove it from its source.

For example, if the map completes successfully, remove the messages from the message queue.

OnFailure

The **OnFailure** setting specifies the location of the source data if the map, burst, or card does not successfully complete.

Note: The exact meaning and availability of the options for this setting is adapter-specific. See your adapter documentation for more information.

The **OnFailure** setting works with the **Scope** setting, which defines when to apply the action specified with the **OnFailure** setting.

Value Description

Rollback

Any changes made during map processing are rolled back and the original state of the data is restored.

If the map, burst, or card does not complete successfully, rollback any changes that were made to the source data during map execution.

For example, if the source of the input data was a stored procedure in a database that changed the contents of one or more tables, such as with an INSERT or UPDATE statement, if the map fails, the **Rollback** option would restore the affected tables to their original state.

Commit

If the map, burst, or card does not complete successfully, commit any changes that were made to the source data during map execution.

If the map fails, send the message produced for the output card to the specified message queue.

For example, if the source of the data was a message on a message queue and the message is removed from the source message queue, regardless of whether the map succeeds or not.

Scope

The **Scope** card setting tells the source or target adapter at what point during map execution to apply **OnSuccess** and **OnFailure** actions. The default setting is **Map**.

Value Description

Map Apply the **OnSuccess** and **OnFailure** settings based on the success or failure of the entire map execution (which may be one or more bursts).

Burst Apply the **OnSuccess** and **OnFailure** settings based on the success or failure of each burst.

Card Apply the **OnSuccess** and **OnFailure** settings based on the success or failure of this input. In this case, when a rule fails or validation fails, the source adapter applies the **OnFailure** action. If validation completes successfully for the input card, the source adapter applies the **OnSuccess** action.

FetchAs and Scope actions

The **FetchAs** setting, as well as the **Scope** setting, affects when an **OnSuccess** or **OnFailure** action occurs.

If FetchAs is:	And the Scope is:		
	Map	Burst	Card
Integral	After all bursts	After the first burst	After input validation for that input for the first burst
Burst	After all bursts	After each burst	After input validation for that input for each burst

If an input **FetchAs** is integral, any failure action causes the entire map to fail.

TargetRule output card settings

The **TargetRule** settings define how the output data is routed. The data object specified in the output card defines what data is output.

To define card-specific output data routing, define the **TargetRule** settings for each output card. **TargetRule** settings specify where to send the data, and what to do when an error occurs. An output card represents an output target. The content of an output is a *data object* such as a table of customer records in a database, or a message that contains a business transaction, or a file of records.

The content of an output card is defined with the **CardName**, **TypeTree** and **Type** settings. These settings define what the data is, but does not specify how to route the data. The **Target**, **Retry**, and **Scope** card settings are some of the **TargetRule** settings that define how the output data is routed for each output card.

To define TargetRule settings:

1. Select the output card.
2. From the **Card** menu, choose **Edit**.
The Edit Output Card dialog box displays.
3. Enter desired values for **TargetRule** settings.

TargetRule Setting overrides

Output card settings are stored in the compiled map. It is possible to override the setting for each option from the both the Command Server and the Integration Flow Designer, which are supplied with the Design Studio.

Execution commands can be used to override the map settings or card settings compiled into the compiled map when a map is run. See the execution commands list for a complete description of the override commands.

The **TargetRule** card settings and their override execution commands are listed in the table below:

TargetRule Card Setting	Override Execution Command
Backup	Not available
Target	Available (-OA), (-OD), (-OE), (-OF), (-OM)
OnSuccess	Available (-O...X)
OnFailure	Available (-O...B)
Retry	Available (-O...Rcount:interval)
Warnings	Not available
Scope	Not available

PUT

The **PUT** settings define all of the output data settings.

FilePath

When **Target = File**, the **FilePath** setting specifies the name and path of the output data file. Enter the filename or browse your file structure to select the output file.

Target > Command

The **Command** setting provides the adapter-specific commands to connect to the data target and process the output data.

For instance, if the output data is being put into a file, the **Command** specifies the name of the output file. If the output data is being sent to a message queue, the **Command** might include the name of the message queue and any other setting required to connect to the message queue and put the message there.

The command syntax for the source and target adapter commands is detailed in the adapter-specific documentation.

DatabaseQueryFile

When **Target = Database**, the **DatabaseQueryFile** settings are used to specify the database specific settings: the name of the desired database/query file, the database name, and the query.

DatabaseQueryFile > File

Specify the database/query file (.mdq) which contains the desired target database and query definition.

DatabaseQueryFile > Database

Specify the database that contains the desired query definition.

DatabaseQueryFile > Table

Specify the database/query file (.mdq) which contains the desired target database and query definition.

Transaction

The **Transaction** settings include the output data **OnSuccess**, **OnFailure**, and **Scope** settings.

OnSuccess

The **OnSuccess** setting specifies the location of the target data when a map, a burst, or a card completes successfully. The availability of the **OnSuccess** setting is adapter specific. The default value is **Create**.

The **OnSuccess** setting works in conjunction with the **Scope** setting, which defines when to apply the action specified with the **OnSuccess** setting.

Value Description

Create Output data should be sent by the target adapter to its target. For example, if the map completes successfully, the message adapter should put the message on the specified queue.

CreateOnContent

Output data should be sent only if content exists. For example, a message adapter should put the message produced on the specified message queue. If no data was produced, no message should be sent.

!Create

Regardless of whether or not data content is produced, output data is not to be sent to its target. This option is available for temporary data storage as a map runs.

Append

The output data should be appended to an existing data file. If the data file does not exist, it should be created.

OnFailure

The **OnFailure** setting specifies the location of the target data if the map, burst, or card does not complete successfully.

The exact meaning and availability of the options for this setting are adapter specific. The default value is **Rollback**.

The **OnFailure** setting works in conjunction with the **Scope** setting, which defines when to apply the action specified with the **OnFailure** setting.

Value Description

Rollback

Any changes made during map processing are rolled-back and the original state of the data is restored. If the map, burst, or card does not complete successfully, reverses any changes made to the target data during map execution.

Commit

If the map, burst, or card does not complete successfully, commit any changes that were made to the target data during map execution. If the map fails, send the message produced for the output card to the specified message queue.

Scope

The **Scope** card setting tells the source or target adapter at what point during map execution to apply **OnSuccess** and **OnFailure** actions. The default setting is **Map**.

Value Description

Map Apply the **OnSuccess** and **OnFailure** settings based on the success or failure of the entire map execution (which may be one or more bursts).

Burst Apply the **OnSuccess** and **OnFailure** settings based on the success or failure of each burst.

Card Apply the **OnSuccess** and **OnFailure** settings based on the success or failure of this output. In this case, when a rule fails or validation fails, the target adapter applies the **OnFailure** action. If validation completes successfully for the output card, the target adapter applies the **OnSuccess** action.

Input and output card settings

The **Retry** and the **Backup** settings appear on both input and output cards.

Retry

The **Retry** settings are defined for each input and output card. The **Retry** setting is compiled into the map, but may be overwritten by the execution settings defined in the Integration Flow Designer and the **AdapterRetry** (-Y) execution command.

If adapter errors occur (for example, failure to connect) and **Retry Switch = On**, the adapter process is retried at the interval specified with the **Interval** value up to as many times as defined with the **MaxAttempts** setting.

The **Retry** settings specify the number of times and the interval at which to attempt access an unavailable source or target.

Define the **Retry** settings for each input (or output) card as required.

Retry > Switch

The **Switch** setting determines whether **Retry** for the specific adapter is **ON** or **OFF**. The default value of the **Switch** setting is **OFF**.

Value	Description
-------	-------------

OFF	Retry is inactive. No attempt is made to access an unavailable source or target.
-----	---

ON	Retry is active. Attempts are made to access an unavailable source or target for the number of attempts specified with MaxAttempts and at the time interval specified with Interval .
----	--

Retry > MaxAttempts

The **MaxAttempts** setting specifies the maximum number of times to try access to a source or target for a map. The default value for the **MaxAttempts** setting is 0.

Value	Description
-------	-------------

nnnnn	
-------	--

A number that represents the maximum number of times an attempt is made to access the source or target for a map. After the maximum number of attempts has been attempted without access, the map returns a return code and message. Valid entries are integers from 0 to 65535.

Retry > Interval

The **Interval** setting specifies the interval (in seconds) at which each attempt to access the source or target is made. The default value is 0.

Value	Description
-------	-------------

- n A number that represents the number of seconds at which each attempt to access the source or target is made.

DocumentVerification

Use the **DocumentVerification** setting for the supplemental parsing (validation) of XML documents during the standard data validation process. In addition to the standard data validation process that takes place when a map runs, enabling this setting instigates an external program to validate XML data according to the schema or DTD selected.

You can choose the specific situations in which the verification process takes place from the map input and output cards, when the **Document Type** property in the associated type tree is set to **XML**.

Using DocumentVerification

To use this feature, in the Type Designer, the type tree object associated with the **Document Type** should have the following properties set:

- The **Document Type** property must be set to **XML**.
- The **Document Type Metadata** value must be set to either **DTD** or **Schema**.
- A DTD or schema file location must be specified (**Document Type Location**).

In the Map Designer, the following should apply in the map input card:

- The **Schema Type** is set to the object you associated with the **Document Type** above.
- The **SourceRule GET** → **Source** → **DocumentVerification** setting is set to **Always**, **OnSuccess**, or **OnFailure**.

Based on these settings, when the map runs, an external parser (document verification program) validates the XML document. If there are no errors, the map completes successfully. If there are errors, the map fails and the error information is written to the map trace file if it is enabled.

Because this setting is at the card level, you can set different options for different input and output cards. The available options are listed in the following table. The default setting is **Never**.

The supported formats are XML with a DTD and XML with an XML Schema.

Value	Description
-------	-------------

Never	The option is turned off, meaning that only the standard data validation process takes place.
--------------	---

Always	Validates an XML document whenever an instance of an object with XML as the Document Type property is found at the card level. This takes place in addition to the standard data validation process by using an external validation program.
---------------	--

OnSuccess	Validates an XML document when an instance of an object has XML as the Document Type property at the card level and is considered valid during the standard data validation process. This supplemental validation takes place in addition to the standard data validation process by using an external validation program.
------------------	--

OnFailure

Validates an XML document only when an instance of an object has **XML** as the **Document Type** property at the card level and fails validation during the standard data validation process. This supplemental validation takes place in addition to the standard data validation process by using an external validation program.

Using DocumentVerification for output data

For output data, the **DocumentVerification** option only works if the **MapAudit SettingsAudit → Data** setting is enabled for the output. This is because output data is only validated (by means of the standard validation process) if the **MapAudit SettingsAudit → Data** map setting for the output is enabled and the output item is specified under **Data Audit Settings** in the **Organizer** as shown:

If the **MapAudit SettingsAudit → Data** map setting is enabled for an output card, standard data validation occurs during the audit process, and the external verification program can run when specified.

Backup settings

The **Backup** settings, when enabled, control when, where, and how the data for input and output cards should be copied to a specified backup file.

You can define the **Backup** settings for each input or output card as needed.

Defining backup settings

The **Backup** settings are defined at the card level. Define the **Backup** setting for each input or output card as required. Expand the **Backup** setting to define the **Switch**, **When**, and **BackupLocation**, which includes the directory and filename values.

To define the Backup settings:

1. From the **Edit Input** or **Output Card** dialog box, locate the **Backup** setting and set the **Switch** to **ON**.
2. Use the **When** setting to define under what circumstances a backup file is created. Select **Always** or **OnError**.
3. Under the **BackupLocation Directory** setting, specify if you want the backup data file written to the **Map** directory or to a **Custom** directory of your choice. If you selected **Custom**, the **Value** field appears under **Directory**. If you selected **Map**, proceed to the next step.
4. In the **Value** field, browse your file structure and select a directory for the backup file.
5. For the **FileName** setting, select **Custom** to create a filename, or **Unique** for a system-generated file name. If you selected **Custom**, the **Action** and **Value** fields appear.
6. For the **Action** setting, specify how data should be written to the backup file. Choose **Create** to create a new file upon map execution, which can overwrite an existing backup file with the same name, or **Append** to append to an existing file, creating it if it doesn't already exist.
7. In the **Value** field, enter a name for the custom file.

Backup > Switch

Use the **Switch** setting to activate the **Backup** feature.

Value	Description
-------	-------------

OFF	Data for the card is not copied to a backup file.
------------	---

ON	
-----------	--

	Data for the card is copied to a backup file based on the subsetting values selected.
--	---

Backup > When

The **When** setting specifies when a backup file is created. The default setting is **Always**.

Value	Description
-------	-------------

Always	
---------------	--

	Always create a backup of the data.
--	-------------------------------------

OnError	
----------------	--

	Create a backup only if a burst fails to complete successfully.
--	---

Backup > BackupLocation

The **BackupLocation** value indicates that backup data is copied to a file. There are subsettings that enable you to select information about the file, such as where and when it is created.

BackupLocation > Directory

BackupLocation Directory enables you to specify the directory into which the backup file is written. The file is written to the **Map** directory by default.

Value	Description
-------	-------------

Map	Creates the backup file in the map directory.
------------	---

Custom	
---------------	--

	Specifies the directory location for the backup file.
--	---

	When you select Custom , a Value field appears where you can enter the file path.
--	---

Directory > Value

If you selected the **Custom** value for the **BackupLocation Directory** setting, type the file path in this field or browse for a file.

BackupLocation > Filename

The **BackupLocation Filename** setting is used to specify the type of naming scheme used for the backup file. **Custom** is the default setting.

Value	Description
-------	-------------

Custom

Specifies a custom name for the backup file. When you select **Custom**, the **Value** field appears into which you can enter the custom file name.

Unique

Generates a unique audit log name in the following format:

Mer_< *mapname*> _<*process_key*>_<*map_counter*>.[*extension*]

Run_< *mapname* >_<*process_key*>_<*map_counter*>.[*extension*]

The prefix "Mer" is used with top-level maps while "Run" is used for run maps.

- *mapname* is the executable map name.
- *process_key* is a unique, per process value.
- *map_counter* is a unique, per map instance value (within the same process).
- *extension* is the backup filename extension, which is **.Inn.bak** (for input cards) or **.Onn.bak** (for output cards), where *nn* represents the card number.

BackupLocation > FileName > Action

The **BackupFileAction** setting specifies how data should be written to the backup file. **Create** is the default setting.

Value Description

Create Creates the backup file at the start of map execution. If a file exists with the same file path, it is overwritten. If more than one burst is used, data will be appended to the same backup file.

Append

If a file exists with the same file path, the data is appended to the existing file. Otherwise, a new file is created. If more than one burst occurs, data for each burst is appended to that same backup file.

BackupLocation > FileName > Value

If the **BackupLocation Filename** setting is **Custom**, enter the file name in this field.

Chapter 7. Map rules

Map rules are used in output cards of a map to define how output data is built. Map rules are required for output data generation. If there is no map rule, there is no output data. When no output is desired for a given data object, a map rule of =NONE is required.

A map is not complete until each rule cell has a map rule, or appears dimmed (the map rule is unavailable). Empty rule cells generate a build error when the map is built.

A map specifies how to generate a particular output data object, or multiple independent objects, each of a particular type. Each output card has a rule column in which you enter map rules. Color-coding of map rules provides visual ease of use in formulating and understanding rules. When color-coding is enabled, syntax errors in map rules are underlined.

Map rules are validated only when that map rule is changed.

Map rules overview

Map rules are used to:

- Convert an object in the input data from one value to another value in the output data
- Perform conditional logic
- Execute another map to produce file with specific contents

A map rule begins with an equal sign (=), followed by an expression that evaluates to data. The maximum length of a map rule is 32K characters (including non-printable characters).

Non-printable characters are not displayed in the Rule Bar, but display in the rule cell on the To window as o.

Color coding of map rules

To aid in readability, map rules can be colored with user-configured colors. When color-coding is enabled, each element of a map rule can be configured with a different color.

The elements of map rules are categorized to allow color specification by category. Colors are assigned to:

- Unknown
- Type Syntax
- Name
- Number
- Map Name
- Reserved
- Literal
- Comment

Enabling color coding for map rules

Color-coding is enabled as a configuration option on the Options dialog box.

To enable color coding for map rules:

1. From the **Tools** menu, choose **Options**.
2. Select **Rule Bar**.
3. Click the **Color Coding** tab.
4. Enable the **Use color coding** check box.

Configuring colors for map rules

Enable color-coding and select a color for each category of map rule elements.

To configure map rule colors

1. From the **Tools** menu, choose **Options**.
2. Select **Rule Bar**.
3. For the Color Specifications on the **Coding** tab, select a category.
4. Display the color palette and select a color.
5. Click **OK**.

To disable color coding

Disable the **Use color coding** check box.

Map rules as expressions

Each map rule begins with an equal sign (=), followed by an expression.

A map rule is an expression that evaluates to data. An expression is any valid combination of literals, data object names, operators, functions and map names. The expression in a map rule generates the desired data. For example:

- The expression `Name Field:Input` evaluates to the value of the data object `Name`, which is a subtype of `Field`, which is a component of the data object `Input`.
- The expression `500` evaluates to a data object with a value of 500.
- The expression `COUNT (Record:OrderFile)` evaluates to a data object that has a numeric value equal to the count of the `Record` objects in the data object `OrderFile`.

A map rule has a maximum length of 32K characters.

Some examples of simple expressions are shown in the following table.

Expression

This expression uses:

"ABC Company"
a literal

City Field:Record
a data object name

City Field:Record + State Field:Record + ZipCode Field:Record
operators (plus signs) and data object names

UPPERCASE (City Field:Record)
the `UPPERCASE` function and a data object name

RecordMap (Record:InputFile)

a map name (RecordMap) and data object names

An expression can be complex and have simpler expressions nested within it. Examples of more complex expressions using nested expressions are shown below:

```
IF (Qualifier = "ST", Address, NONE)
LOOKUP (ProductCode:LookupList, ItemCode:Data =
ItemCode:LookupList)
EXTRACT (Record:Input, Company Field:Record:Input = "My
Company")
COUNT (EXTRACT (Row:Input, Company Column:Row:Input = "My
Company"))
```

These examples use the functions IF, LOOKUP, COUNT, and EXTRACT. These functions are specific to the WebSphere Transformation Extender suite of products. Functions have their own syntax. For more information on functions, see the Functions and Expressions documentation.

Map rules define how to generate data objects

The object of an output card may be composed of components and partitions. Each object on the output card has a corresponding rule cell, where map rules that transform that data are entered.

A map rule evaluates to a particular data object. The data object is either a group type or an item type. The following rule evaluates to the data object **Record**, which is a group:

```
= LOOKUP (Record:Input, Company:Record:Input = "ABC Printing")
```

A map rule for a data object must evaluate to a data object with the same subclass. Objects with an Item Subclass of **Date & Time** must evaluate to an object with an Item Subclass of **Date & Time**.

For example, the output item ShipDate has the Item Subclass of **Date & Time**. The following map rule is valid when the data object ShipToDate also has an Item Subclass of **Date & Time**:

```
ShipDate = ShipToDate:P0:Input
```

The output CompanyName is defined with an Item Subclass of Text. The following map rule:

```
CompanyName = "Pop's Pizza Co."
```

assigns the value of a text literal to the output.

A text literal is text enclosed in double quotes and is interpreted as a text item. A rule that evaluates to something other than text, for example, CompanyName = 24 would be invalid.

To view the type properties of a data object in a card, right-click the type in the input or output card and choose **Properties** from the context menu. If the Properties window is shown, select the map rule to update the Properties window to show the properties of the object.

A map rule on a group type must evaluate to a data object of the same group. For example, the following map rule on the output group **Record** must evaluate to the group type **Record**:

```
Record = LOOKUP (Record:Input, Name:Record:Input = "Kathy")
```

The map rule is for the group type **Record**, so it must evaluate to the type **Record**. The rule shown above does evaluate to **Record** because the LOOKUP function returns a **Record**. The rule would be invalid, for example, if it evaluated to the group **Invoice**.

Map rules are evaluated independently in sequential order

Each map rule is evaluated independently from all other rules. Each output object with a rule is created independently from the others.

Map rules are evaluated in sequential order. When a map is executed, the first rule of the first output card is evaluated. Next, the second rule of the first output card is evaluated, and so on, until the last rule of the first output card. Then, the first rule of the second output card is evaluated, and so on, to the last rule of the last output card.

Unavailable rule cells

The output of a map produces each output card object. To produce a card object, the required objects that compose the card object must be produced. There is a rule cell for each object within the card object.

If an output object is an item, you must supply a rule for it. If an output object is a group, you have two options. Either you can enter a rule for the object as a whole, or you can expand the view of that object in the output card and enter rules for each data object.

Defining how each component of the output object as a whole must be produced is the equivalent of defining how to create the entire output object.

If map rules are entered for each component of the output object, map rules may not be entered for the entire output object. When you expand the entire output object, notice that the rule cell next to the output object is automatically unavailable (grayed out). Entering a rule for the entire output object would be redundant, because the individual components of the output object already have map rules.

When you expand a group that has a component range maximum greater than 1, the rule cells on the components of that group are unavailable.

Map rule for the entire output object

You can enter a rule for **Product Record**.

Or expand **Summary** and enter a rule for each component: **Explanation** and **TotalProducts**.

Map rule for each component

You can enter map rules for each component of **Summary** or enter a rule for the entire **Product Record** object.

Entering map rules

Enter map rules by:

- Typing
- Dragging a data object from an input card
- Copying and pasting a data object from an input card
- Dragging a map name from the Navigator
- Inserting a function
- Dragging an item restriction from the Properties window into the rule bar

To enter an object name in a rule, drag the object from the input card into the rule. It is easier than typing the name and reduces the possibility of entry errors.

Inserting a function into a map rule

Functions can be inserted into map rules.

To insert a function into a map rule:

1. Select the rule cell where you want to insert the function.
2. If necessary, click in the rule bar in the specific place where you want to insert the function.
3. From the **Rules** menu, choose **Insert Function**.
The Insert Function dialog box is displayed.
4. If you know the function category, select the **Category**.
5. Select the function you want to insert.
6. Click **OK**.

The function is inserted into the rule bar.

7. Insert another function or click **Close**.

In the Insert Function dialog box, Help is available for each function. With current focus on the desired function, press F1 to view help on that function.

Inserting a symbol into a map rule

Symbols are used to indicate non-printable characters. Symbols in map rules can be entered using the Symbols dialog box. Symbols can also be entered using the angle brackets around the decimal value.

For example, the carriage return/line feed symbols can be inserted from the symbols dialog box or entered in angle brackets.

Using the Symbols dialog box, the carriage return/line feed appears in literal quotation marks as "<CR><LF>".

Alternatively, enter the hexadecimal values in double angle brackets <<0D>><<0A>>.

To insert a symbol:

1. Place your cursor where you want the symbol to be entered.
2. Browse to display the Symbols dialog box.
3. Select the desired symbol and click **Insert**; or, double-click on the desired symbol. (In either case, the symbol appears in the **Value** field.)

4. Click **OK**.

The symbol appears within literal quotation marks in the map rule.

Entering a restriction into a map rule

If you want to enter a map rule that includes those restriction values, you can drag the restrictions from the Properties window into the rule bar.

To enter a Restriction in a map rule:

1. Select the rule cell where you want to insert the restriction.
2. Right-click on the item whose restriction(s) you want to enter, and choose **Properties**.
3. Click the **Restrictions** tab.
4. Drag the restriction you want into the rule bar.

The restriction appears in the rule bar.

Dragging a restriction for an item with an **Item Subclass** of **Text** causes the restriction to be automatically enclosed in quotation marks in the rule.

Generating no output

Each map rule cell requires a map rule. For map rules for data objects for which you do not want to generate output, enter the following:

= NONE

The reserved word **NONE** specifies to generate none of the given output.

You don't have to type in the rule = **NONE**. The command **Insert NONE if Empty** puts this rule into the empty rule cells on the active output card, including those empty cells that cannot be seen by the current view of the card.

To insert = **NONE** in every empty rule cell:

1. From the **Rules** menu, choose **Insert NONE if Empty**.

Only empty rule cells that are showing when you scroll up and down in the output card will be filled. To enter = **NONE** in all empty cells, including the ones that are nested beneath unavailable cells, use the **Show All** command on the **Rules** menu and then choose **Insert NONE if Empty**.

Special characters in map rules

To enter a Hexadecimal value of a special character in a map rule, enter the Hex value for the characters in double brackets (open and close carets).

For example, to enter the text value <WSP>, enter the Hex value for the open caret <<3C>>, and then the literal values for the rest: **WSP**

<<3C>>WSP>

Referencing an output in a map rule

To enter the object name of an output data object, drag it to the Rule Bar just as you do with an input data object. Output data is built from top to bottom, beginning at the first output card.

You can map an output to another output, as long as the output you are mapping *from* is evaluated before the output you are mapping *to*.

You can reference an output in the same card or another card. You can reference a data object from another output card, as long as that output card is evaluated *before* the output card containing the map rule.

To reference an output in an earlier output card:

1. To show two views of the output card, drag any card from the To window to the main window.
2. View and select the destination rule cell where you want to enter the object name into the map rule.
3. In the other view of the To window, view the output card that contains the data object you want to enter.
4. Drag the object into the destination rule bar.

Comments in map rules

You can add inline comments to map rules. Comments do not affect how map rules are evaluated. A comment may appear anywhere in a rule, as long as it does not separate object names.

A comment is identified with the start characters `/*`. While the end characters `*/` can be present, they are not required. If the end characters are not present, everything up to the end of that rule is the comment.

The following map rule has two comments, which are indicated by blue text:

```
= EXTRACT (Record:Customers /*Extract the Records */,  
State Field.:Customers = "FL" /* Get only the customers in  
Florida */ )
```

Using comments for map rules, or portions of map rules, is a convenient way to retain complex and lengthy map rules for future use without deleting them.

Syntax errors in map rules

Syntax errors are reported when the map rule is committed. The default "commit" key is the Enter key. Pressing Enter on a map rule commits that rule and reports any syntax errors.

The cursor is automatically positioned at the first syntax error. When color coding is enabled, syntax errors can be configured to display with underline.

Data object names in map rules

The following examples show different kinds of data object names that can appear in a map rule. The following is a partial list of the possible data object names. For a detailed list of object names, see the Functions and Expressions documentation.

Data Object	Appears in map rules as:	Description
Card names	OrderFile	User-defined name of input or output card A unique name for each card in the map provides the ability to distinguish between card objects that may be used in more than one card.

Data Object	Appears in map rules as:	Description
Component	Record:OrderFile Quantity:Record:OrderFile	Separated by colon (:) Interpret as "of": " Quantity of Record of OrderFile " Quantity is a component of Record . Record is a component of OrderFile .
Index	Note[3]:Report	Specific occurrence appears in square brackets [<i>n</i>] immediately after the type A particular occurrence of a data object indicated by the index
Partition	Customer<>Record	Separated by < > When a type is partitioned, its subtypes, or partitions, appear in the card with < >.
Shortened data object name	Company Field:::Input	A period (.) is used as an abbreviation of ellipses (...) The shortest object name that is different from all other object names in the same map is used. Some object names may not include ellipses, because the shortest unique name is the entire name.

Card names in map rules

A map rule can see the entire card data object. For example, if an input card name is **OrderFile**, the name for the entire card object is **OrderFile**.

The name of the card appears as the top data object of the card. In the example shown below, the card name is **OrderFile**. This name appears last in each object name:

Record:OrderFile

Quantity:Record:OrderFile

A unique name for each card in the map provides the ability to distinguish between card objects.

For example, if two input cards are defined by the same type, these two input cards may exist with the same type, but their card names are different. If the name of the first card is **Input1** and the second input card is **Input2**. If a component **Record** exists as data objects in both these cards, the difference is distinguishable **Record** in one input and **Record** in the other because the **Record** types in the different cards have different names **Record:Input1** and **Record:Input2**.

Component names in map rules

A component in a data object name is indicated by a colon (:). The colon is the separator between a data object and the group or category type it is a part of (or the card name it is contained in).

For example, **Record** is a component of the data object **OrderFile**. The object name for **Record** is **Record:OrderFile**.

When you see a colon (:), you can read it as "of". For example, the name of the component **Quantity** in the input card is **Quantity:Record:OrderFile**. You can read the data object name for **Quantity** as "The **Quantity** of **Record** of **OrderFile**".

Index in map rules

An object name can refer to a particular occurrence of a data object indicated by the index.

For example, the third **Note** in a series of **Note(s)** has an index of 3. In a data object name, an index appears in square brackets immediately after the type being indexed. The indexed third **Note** of **Report** is **Note[3]:Report**.

Partition in map rules

When a type is partitioned, its subtypes or partitions appear in the card.

In a map rule, a partition is referenced by the symbol <>.

For example, the partition **CallRecord** of the group type **SetofCallRecords** is **CallRecord<>SetofCallRecords**.

Think of a partition as a particular kind of the partitioned type. For example, in the name **Detail<>Record**, you can think of it as a **Detail** kind of **Record**.

An object name includes component references and partition references. For example, the partition **Back** of **OrderRecord**, on the input card **File**, appears in a map rule as **Store Field:CallRecord<>OrderRecord:File**

Use ellipses

Enables the **Use Ellipses** display option for subsequent object names dragged to the rule bar. The **Use Ellipses** options causes lengthy object names to display the shortest possible object name.

Rather than having the entire name appear, unique portions of the name are replaced with a period (.) that is used as an abbreviation of ellipses (...).

Complete object name: **Company Field:Record:Input**

Shortened name: **Company Field::Input**

If the **Use Ellipses** option is enabled, the shortest object name that is different from all other object names in the same map is used. Some object names may not include ellipses, because the shortest unique name is the entire name.

The **Use Ellipses** option does not affect object names that already exist in map rules; it only affects object names that are entered after the option is enabled.

Using ellipses to shorten an object name

Use ellipses to have object names in map rules.

To have new object names appear with ellipses:

1. From the **Tools** menu, choose **Options**.
2. Choose **Rule Bar**.
3. Enable the **Use Ellipses** check box.
4. Click **OK**.

The **Use Ellipses** option does not affect object names that already exist in map rules; it only affects object names that are entered after the option is enabled.

Using the toolbar to enable **Use Ellipses** changes the setting in your **Options Rule Bar** utility.

Viewing properties in the Map Designer

In the Map Designer, you can always access information about data objects and functions. You can view the following:

- Properties of a type
- Component rule of a component
- Component attributes of a component
- Inputs and output of a function
- Inputs and output of a functional map by viewing the map rule properties
- Contents of a map rule
- Syntax of a function

Viewing type properties

Properties of a data object are defined in the Type Designer, but can be viewed in the Map Designer. Data object properties are displayed on the **Properties** tab in the properties window. There are two columns on the **Properties** tab: the settings of the data Item or Group column, and the Value column.

To view the properties of a type

1. Right-click the type in a map card.
2. From the context menu, choose **Properties**.
3. Click the **Properties** tab to view type properties.

You can also view attributes, component rules, and restrictions by clicking the **Attributes**, **Rule**, and **Restrictions** tabs.

To update the view on the Properties tab

1. With the properties window open, update the view by clicking the data object in the **Output** or **Rule** column of the card.
2. The title bar of the properties window displays the data object name.

Viewing component rules

Component rules are displayed on the **Rule** tab in the properties window.

To view the component rule of a component

1. Right-click the component in a map card.
2. From the context menu, choose **Properties**.
3. Click the **Rule** tab.

To update the view on the Rule tab

When the properties window is already open, update the view by clicking the data object in the map card.

Viewing component attributes

Three attributes can be assigned to a component in the type tree: identifier, restart, and sized. These attributes are viewed in the Map Designer on the **Attributes** tab of the **Properties** window.

To view the attributes of a component

1. Right-click the component in a map card.
2. From the context menu, choose **Properties**.
3. Click the **Attributes** tab.

To update the view on the Attributes tab

1. When the properties window is already open, update the view by clicking the data object in the map card.
2. The title bar of the properties window displays the data object name.

Viewing restrictions of an item

Item restrictions and values are entered in the Type Designer and can be viewed in the Map Designer. Item restrictions are viewed on the **Restrictions** tab of the properties window.

To view restrictions of an item

1. Right-click the item in the map card.
2. From the context menu, choose **Properties**.
3. Click the **Restrictions** tab.

To update the view on the Restrictions tab

1. When the properties window is already open, update the view by clicking the data object in the map card.
2. The title bar of the properties window displays the data object name.

Viewing function syntax

You can view the syntax of a function in the Insert Function dialog box.

To view function syntax:

1. Click in any rule cell in an output card to enable the Insert Function command.
2. In the Insert Function dialog box, select the function whose syntax you want to view.

The syntax of the function with the inputs and outputs appears in the **Description** pane of the Insert Function dialog box.

Viewing map rule properties

You can find out information on everything that appears in a map rule. The inputs and output of a functional map are in the map rule properties.

To view the properties of a map rule:

1. In the output card, right-click the map rule and choose **Properties** from the context menu.
2. On the left side, click to select an object, function, or functional map.
The **Properties** pane on the right side displays the description, syntax, or properties of the active selection.

Viewing functional map inputs and outputs

The inputs and output of a functional map are in the map rule properties.

Opening the type tree

You can edit a type when you are designing maps in the Map Designer. The group or item window and the Properties window for the selected type are opened.

To open a type tree:

1. In an input or output card, right-click the type whose tree you want to open.
2. From the context menu, choose **Open Type Tree**.

The selected object window is open in the Type Designer.

You can view the type properties by right-clicking the type and selecting **Properties** from the context menu.

Formatting a map rule

Some map rules may be lengthy. The map rule text wraps in the rule bar. You can create a new line in the rule bar where you want one.

Creating a new line in the rule bar depends on what key is used as the "commit" key (the default is the Enter key). New lines may not be added in literal text strings.

Extra spaces in a map rule may be added to improve readability.

Spaces cannot be added within an object name or function name.

To create a new line in the rule bar if Enter is the Commit key

1. Place the cursor where you want to create a new line in the rule bar.
2. Press **Ctrl Enter**.

To create a new line in the rule bar if Tab is the Commit key

1. Place the cursor where you want to create a new line in the rule bar.
2. Press **Enter**.

Non-printable characters are not displayed in the Rule Bar, but display in the rule cell on the To window as o. A new line in the rule bar displays as **oo** in the rule cell.

Deleting the current selection in map rules

Delete any selected text in map rules or fields.

To delete the current selection:

1. Select the text to be deleted.
2. Right-click and select **Delete** from the context menu.

Chapter 8. Formulating map rules

Implement map rules to build specific output results.

Generating no output

If an output is optional, you may not want to generate the output for that data object. In situations in which you do not want to generate output for a data object on an output card, enter a special rule. The rule for generating none of an output is simply:

= NONE

The word NONE is case-sensitive.

Text literals as NONE

The use of "" as text literals in a map rule evaluates to "none". For example, the following two rules are equivalent:

= EXTRACT (Record:Input, Store:Record:Input = "")

= EXTRACT (Record:Input, Store:Record:Input = NONE)

When input data is missing

If you map an input to an output, but the input is missing, either the output will have no content or it will not exist. The existence of the output depends on how the output is defined.

When portions of an input series are missing

When portions of an input series are missing, the output to which it is mapped built according to the **Track** property of the type in the Type Designer. The **Track** property indicates whether to only track the components that have content (**Track = Content**) or all components, including those that do not have content (**Track = Places**).

If both the input data object and the output data object have the **Track = Places** property, the place holder is generated on output, even if there is no data. Example output data with a missing third item would be:

YES,YES,,YES

If the input data object has the **Track = Content** property, the empty occurrences of that data object are not mapped and the empty occurrences are not built. Example output data with a missing third item would be:

YES,YES,YES

When an output evaluates to NONE

If an output object evaluates to "none", the information appearing in the output depends on how the output has been defined. The following table defines how output is build when the output occurrence evaluates to "none":

Output is defined in this manner	Context	Output is built in this manner
Required item	Always	The item's initiator and terminator are built.
	Occurs within a fixed explicit group	If a special value for NONE has been assigned, NONE is built as the special value. Otherwise, NONE is built as the pad characters for the Padded to Length specified.
	Occurs within a delimited explicit group	The group's delimiter appears in the output. If a special value for NONE has been assigned, NONE is built as the special value. Or, if a pad property is specified as applied in any context, NONE is built. Otherwise, NONE is not built.
	Occurs within an implicit group	If the group has a delimiter, the group's delimiter appears in the output. If a special value for NONE has been assigned, NONE is built as the special value. Or, if a pad property is specified as applied in any context, NONE is built.
Optional item	Always	The item's initiator and terminator are not built.
	Occurs within a fixed explicit group	If a special value for NONE has been assigned, NONE is built as the special value. Otherwise, NONE is built as all pad characters for the specified length.
	Occurs within a delimited explicit group	The group's delimiter appears in the output if data with the same delimiter follows.
	Occurs within an implicit group	No output is built.
Required group	Always	The group's initiator and terminator are built. Required components are built.
Optional group	Always	The group's initiator and terminator are not built.
	Occurs within a fixed explicit group	Each component is built, down to its items, building NONE for each item according to the special value for NONE and pad property specifications.
	Occurs within a delimited explicit group	The group's delimiter appears if data with the same delimiter follows.
	Occurs within an implicit group	No output is built.

An item within a fixed group must have the **Padded to = Fixed Size** property specified, or the size minimum must equal the size maximum. If an item is required, and content is specified as required, an output invalid message appears in the summary trace file.

Mapping an input item to an output item

You might want an output item to be equal to some input item. In that case, drag the input item into the rule cell of the output item.

Automatic item conversions

When mapping an item to an item and the input items have the same subclass as the output items, the item type properties of the input item are automatically used as the item type properties of the output item.

For example, an item type is defined according to the following:

Item Subclass = Number

Presentation = Decimal with a required decimal separator

Pad = Yes

Padded to = Fixed Size

Padded to > Length = 7 bytes

Pad > Value = 0

Pad > Justify = Right

If an input type is defined as a decimal number, with no pad character, no separator, and two decimal places, the following table shows an example of input and output types of an item type definition.

Property	Input Type	Output Type
Presentation	Decimal	Decimal
Separators	No	Yes
Places > Decimal Min	2	2
Pad > Value	(none)	0
Padded to	(none)	7
Justify	(none)	Right

The input data is automatically converted to the format of the output data object. The output data will be in decimal format, right justified, padded to 7 bytes with a pad character of 0.

An input item mapped to an output item must have the same **Item Subclass**. For example, both items must have an **Item Subclass = Number** or both must have an **Item Subclass = Text**. To convert the **Item Subclass** of an item, use the conversion functions, such as `NUMBERTOTEXT`, `TEXTTODATE`, and so forth.

Intermediate item conversions

When using an item with an **Item Subclass** of **Number** or **Date & Time** as a text argument, use the `TEXT` function to convert that item to a text form.

Concatenating text strings

To concatenate text, use the plus symbol (+) between the objects to be concatenated.

For example, the output **Name Field** is defined in the map rule as the input **FirstName Field**, a space, and then the **LastName Field**.

```
=FirstName Field:EmployeeInfo + " " + LastName  
Field:EmployeeInfo
```

Mapping a group to a group

Desired output of an input group may be the same as an input group. If the output group is the same type and is defined in the same type tree as the input group, drag the input group into the map rule cell of the output group. If the input group is a partition of the output group, you can drag the partition into the rule cell of the output group.

If the input group and output group are different types, you cannot drag the input group to the output group. Either expand the output group down to its item components, or use a functional map.

Automatic conversion of syntax items

Syntax items are automatically converted from an input group to the syntax items of the output group. If the delimiters, terminators, and release characters of a data object are variable, these values are defined in the input data stream. To convert these values to other values, you may drag the input group to the output group map rule. The output group is built using the default values or data values for the syntax items.

Mapping to a group with a maximum range of 1

Generally, do not drag a group from the input to the output. This is only possible if the input group and the output group are the exact same group, which is not very common. When the range of a group has a maximum of 1, expand that output group and map to its components.

Outputs with maximum range greater than 1

For outputs with a component range maximum greater than 1 - (for example, (1:10) or (s)), define how many of that particular output to generate. There are two possible situations:

- the number of objects to create is known, or
- the number of objects to create is based on the number of some other object in the input or output data.

When the number of objects to create is known, index that number of output occurrences.

When the number of output objects is based on the number of some other object, create a map rule that maps the output according to the number of occurrences of an input.

Indexing an output

When the number of objects to generate for each map execution is known, index the number of occurrences of that output object. To *index an output* is to generate a specific number of occurrences of that output.

The **Use Ellipses** option must be disabled when indexing. Select **Tools** → **Options** → **Rule Bar** and disable the **Use Ellipses** check box.

To index an output

1. Right-click the output data object you want to index.
2. From the context menu, choose **Add Index**.

An output object is added to the output card. This output object has the index number in square brackets. For example, [1].

The index number identifies the occurrence of the object. The index number [1] indicates the first occurrence. The index number [2] indicates the second occurrence, and so on. After you have indexed an output, you can enter a map rule for that indexed output. If you have indexed a group, you can then expand the group and add map rules at the component level.

Map to the first occurrence of an indexed object by dragging that object from the input card to the map rule of the item on the output card. Define the second occurrence of the output by dragging the desired object from the input card to the map rule of the item on the output card. If you do not want a third indexed data object, enter the map rule = NONE.

To index a group

1. Right-click the group and choose **Add Index** from the context menu.
2. Add an index to the desired groups.
3. Expand those groups, and add map rules at the component level.

To delete an index

Right-click the output of an index and select **Delete Index**.

The indexed occurrence is deleted.

Reordering indexed objects

Indexed occurrences can be rearranged. This is useful if map rules for indexed occurrences need to be moved.

To reorder an indexed object:

Press the **Ctrl** and **Shift** keys and drag the indexed object to the desired position.

All of the map rules nested within that object are moved.

Mapping to multiple occurrences of a group

When an output group has a range with a maximum greater than 1 and you want to map a certain input group to that output group, there are two possibilities:

- If the input group is the *same* as the output group, drag the input object from the input card to the map rule of the object on the output card. The output is created up to its maximum range.
- If the input group is *different* from the output group, use a functional map. To generate occurrences of an output group based on occurrences of input data, use a functional map.

Indexing an Input

When a map rule refers to a particular occurrence of an input, the input must be indexed.

The **Use Ellipses** option must be disabled when indexing. Select **Tools** → **Options** → **Rule Bar** and disable the **Use Ellipses** check box.

To index an input:

In the map rule of the output object, enter the index number in square brackets [] immediately after the input object you are indexing.

Mapping from a floating component type

If an input card data object is a group that has a floating component type, you can map from the floating component type. A floating component must be distinguishable from every component of the group it is assigned to. On input, a floating component can appear after the initiator and after each component. If a floating component is associated with an input data object, it can appear as the first component.

If a type object has content, a floating component is built as the specified literal after the initiator and after each component. If this feature is used, it is up to the user to include a literal that will validate as an object of the type of the floating component.

You cannot map to a floating component in the output. If you have a floating component in the output data to which you want to map, you must define them as components of the output.

To display floating component types on the input card:

1. From the **Tools** menu, choose **Options**.
2. Click **From Window**.
3. Enable the **Show floating component type** check box.
4. Click **OK**.

To map from a floating component type in an input card object, drag the floating component type into the rule cell of the output object on the output card. To map from a nested component of that floating component, drag the nested component.

Mapping from a floating component of a particular component

You can map from a floating component that appears after a particular component of the input card object.

To refer to a floating component after a particular component

1. Select the component whose floating component you want to map.
2. Drag the floating component, or a nested component of the comment, into the output object rule cell.

The object name and the @ symbol indicate the component that the floating component type follows. In this example map rule, **Text Field:Note** is the floating component that follows **Discussion:Presentation**.

Chapter 9. Functional map basics

A functional map is like a subroutine; it maps a portion of data at a time. A functional is a map that is used like a function. It takes one or more input objects and generates one output object. For example, you might have a functional map that maps one **Message** to one row in a database table. Or you might have a functional map that maps one **Header** and one **Detail** to one **ItemRecord**.

The results of the functional map are sent directly to the output card. They are not passed back to the calling rule.

When to use a functional map

The use of functional maps is very common. Almost every executable map created will use at least one functional map.

To map a group in the input to a different group in the output, use a functional map.

For example, use a functional map to map an input row to an output row when the rows are defined differently. Or, use a functional map to map from a file containing many input rows, to generate a file of many output rows with one output row per input row. The first output row would correspond to the first input row, the second output row corresponds to the second input row, and so on.

A map defines how to generate the output data. One important factor to consider in determining when to use a functional map is the presence of an output component with a range of more than one. For example, ranges of (s) or (1:10). The number of this output object to be created is based on the number of some input object.

Another important factor in determining when to use a functional map is when you want to transform the data - mapping from one or more types to a *different* type. In the preceding example of the functional map that maps one **Message** to one row in a database table, the input row and the output row are two different types.

Use a functional map when the number of a certain output group that you want to create is based on the occurrences of some input or output data - and the types are different types.

Similarities of input and output data

Frequently, your input and output data contain very similar information, but the data is arranged differently.

For example, you have a file of invoices from your company's internal application and you want to map it to a file of invoices to send to another company. The input and output invoices are really reflections of one another. The layout of an invoice in the application file may be different from the layout of an invoice in the output file. The important point is that you want to make an output invoice for each input invoice.

Or you may receive health claims in a standard format, and you want to map them to a different standard format. The input claims are embedded in a hierarchical layout where data is not repeated. The output claims may be arranged so that each claim has the same key data repeated: the provider, the patient, and so on. To make an output claim for each input claim, use a functional map. There is a one-to-one relationship between an input claim and an output claim.

A good indication of when to use a functional map is when there is a one-to-one relationship between a group in the input and a different group in the output.

The following example shows a data source of three input records. A functional map provides a way to map one input record to one output record. The functional map defines the first input record that generates the first output record. Next, the second input record generates the second output record. The third input record generates the third output record. The result is three output records.

Using a functional map

The basic steps to follow for using a functional map are as follows:

1. Determine the need for using a functional map.
2. Determine the input argument(s) of the functional map.
3. Enter the map rule that references the functional map.
4. Create the functional map.
5. Create the input card(s).
6. Create the output card(s).
7. Enter map rules in the functional map.

The Functional Map Wizard eliminates Steps 5 and 6.

Example scenario for using a functional map

An example of a functional map is shown below. A data file contains statistics for different kinds of cars:

Marlin	22	2	27	11	2930
Pacer ZSX	17	3	26	11	3350
Wildcat	22	3	18	12	2640
Fantasia DX	17	3	27	15	2830
La Vella	23	2	28	11	2070
Blue Baron	25	2	26	12	2650
Cavalino	20	4	29	16	3250

Each record in the file represents a car. For each car, there are fields for mileage, head room, rear seat, trunk room, and weight. Each field is padded to a specific length. The record is fixed.

The objective is to map this data to a file containing delimited records. The delimited record has only the car name and the weight. In the delimited record, the weight is in decimal format. The desired output data result is:

```
Marlin,29.30
Pacer ZSX,33.50
Wildcat,26.40
Fantasia DX,28.30
La Vella,20.70
Blue Baron,26.50
Cavalino,32.50
```


You want to generate one delimited record for each fixed record in the input file.

Input data	Output data
Marlin 22 2 27 11 2930	Marlin,29.30
Pacer ZSX 17 3 26 11 3350	Pacer ZSX,33.50
Wildcat 22 3 18 12 2640	Wildcat,26.40
Fantasia DX 17 3 27 15 2830	Fantasia DX,28.30
La Vella 23 2 28 11 2070	La Vella,20.70
Blue Baron 25 2 26 12 2650	Blue Baron,26.50
Cavalino 20 4 29 16 3250	Cavalino,32.50

Define the map rule for the output **CarRecord(s)**, with a range of (s). There is an indefinite number of **CarRecord(s)**.

Ask yourself "how many **CarRecord(s)** do I want to generate?" If you want a specific number, the answer would be that number. To generate two **CarRecord(s)**, index two occurrences of **CarRecord**, expand each occurrence, and map to its components. For more information on indexing, see "Indexing an Output".

If the number of **CarRecord(s)** is based on the number of **FixedRecord(s)** in the input, generate one **CarRecord** for each **FixedRecord**. So, the answer to your question of how many **CarRecord(s)** to generate is, "As many as there are **FixedRecord(s)** in the input."

Note: When you see an output group with a range, it is always a good idea to stop and ask yourself, "How many of these do I want to generate?" The answer will either be:

- A specific number - index the output.
or
- The number is based on the number of some input - use a functional map.

You need a mechanism that takes a single **FixedRecord** and creates a single **CarRecord**. To do this, create a functional map, which is similar to a user-defined function. This functional map maps a single **FixedRecord** to a single **CarRecord**. The map is called for every occurrence of a **FixedRecord** in the input. This generates exactly one **CarRecord** for each **FixedRecord**.

For example, if there are 23 **FixedRecord(s)**, 23 **CarRecord(s)** are generated. The first **CarRecord** corresponds to the first **FixedRecord**. The second **CarRecord** corresponds to the second **FixedRecord**, and so on.

Comparing executable and functional maps

There are two kinds of maps: executable maps and functional maps.

Executable Map

An *executable map* is a map responsible for the totality of your inputs and outputs. The sources and targets of an executable maps are entire files, database tables, messages, applications, and so on. Think of an executable map as the main map, the top-level map. Executable maps are compiled and run.

Functional Map

A *functional map* is referenced by another map through a map rule. A functional map maps just a portion of the entire data. Data sources and targets are not specified in the cards of functional maps. Functional maps are not compiled and run.

Note: A given map can be used both as an executable and as a functional map. You may want to test small portions of your data and temporarily use a map you created as a functional map as an executable map on that portion of the data.

If a map is used as a functional map, the input data and output data settings specified in that map are ignored.

Syntax of a functional map expression

Map rules on executable map output cards specify the functional map name. The syntax of a functional map expression is similar to the syntax of a function.

Similar to a function, a functional map has one or more inputs, and one output.

The inputs and the output are data objects.

The syntax of a functional map expression is the following:

```
FunctionalMapName (argument1, argument2,...argumentn)
```

Where *n* is the argument number. The input arguments of a functional map appear between the parentheses in the functional map expression separated by commas.

An input argument of a functional map is any valid expression that evaluates to data, including data objects, functions, and literals. For example, an input argument may be any of the following:

Input argument

Example

data objects

```
FixedRecord:Input
```

functions that evaluate to data

```
EXTRACT (StoreInfo:Inventory, StoreName:Inventory = "Lee  
Furniture")  
INDEX (Record:File)
```

literals

```
"payroll"
```

Each of these expressions evaluates to data. For example, the following expression evaluates to the data object **FixedRecord**:

```
FixedRecord:Input
```

The following expression evaluates to **StoreInfo** data objects:

```
EXTRACT (StoreInfo:Inventory, StoreName:Inventory = "Lee Furniture")
```

The `INDEX (Record:File)` expression evaluates to an integer; the `"payroll"` expression evaluates to text.

For a detailed discussion of functions and expressions, see the Functions and Expressions documentation.

Determining the arguments of a functional map

The input arguments of a functional map are the objects necessary to create *one* occurrence of the output object where the map rule is.

To determine what the input argument(s) of a functional map should be, ask yourself, "What do I need to create this output?" Whatever is necessary in a functional map must be passed to it as an argument. There may be one or more input arguments. The arguments of a functional map can be thought of as triggers. The number of times a functional map is triggered depends on the number of occurrences of the input arguments in the data and the expressions used for each argument.

To determine the input arguments of a functional map, ask yourself what objects are necessary to create *one* occurrence of the output object.

Input argument to a functional map evaluates to NONE

If an occurrence of input argument is missing, the functional map is not evaluated.

A file that has no content will still allow the functional map to execute.

For example, in the following functional map expression the **GroupInfo** object is optional with a component range of (0:3):

```
MakeForm (EntryForm:Input, GroupInfo:Input)
```

If there are no occurrences of **GroupInfo** in the source data object, the specified **MakeForm** functional map is not evaluated.

In the following map rule, the **OrderMap** functional map is specified with the (DetailRecord:Order:Input, SummaryRecord:Order:Input) arguments:

```
OrderMap (DetailRecord:Order:Input, SummaryRecord:Order:Input)
```

The data object **SummaryRecord** has an optional component range of (0:1). For the first **Order**, **SummaryRecord** is present. The functional map **OrderMap** is evaluated. In the second **Order**, **SummaryRecord** is missing. The map **OrderMap** is not evaluated for that second **Order**.

To evaluate the entire **OrderMap** if **SummaryRecord** may be absent from the source data, specify the entire **Order** as an input argument, rather than **SummaryRecord**. Specifying the entire **Order** ensures that **OrderMap** is evaluated, even when **SummaryRecord** is missing. Then, in the functional map **OrderMap**, you can still map from **SummaryRecord**.

Entering the map rule that references the functional map

Map rules that reference functional maps are entered in the rule cells on the output card of the executable map. Map rules belong in the rule cell of the data object that requires additional data processing. The number of times a functional map is triggered depends on the number of occurrences of the input arguments in the data and the expressions used for each argument.

This section explains how to manually create a functional map. You can also use the "Functional Map Wizard".

For the executable map for mapping the car data (see Example Scenario for Using a Functional Map), the executable map is **CarData**. The source data contains fixed records and the goal is to generate an output file containing one delimited record for each fixed record in the source data.

A functional map on the **CarRecord(s)** data object can generate one **CarRecord** for each **FixedRecord**. A suitable name for the functional map can be **RecordMap**, because the functional map maps one record at a time. In the **CarData** executable map, the map rule for the **CarRecord(s)** data object in the output card would contain an equal sign, the name of the functional map, and a set of parentheses:

```
=ListCars ( )
```

To map the fixed record input object to the map rule, drag the **FixedRecord(s)** data object from the input card into the rule cell inside the parentheses. Commit the map rule by pressing the **Enter** key.

A map rule specifies how to generate that certain output. The map rule on **CarRecord(s)** specifies how to create each **CarRecord**. Basically, the map rule specifies "to generate a **CarRecord** data object, use a functional map named **ListCars** on a **FixedRecord** data object." The functional map **RecordMap** uses the **FixedRecord** to create a **CarRecord**.

The functional map **ListCars** can be created before the executable map. The creation order for the map rule or functional map is not relevant. However, an error occurs at build time when a rule includes a nonexistent map name.

Creating a functional map

The functional map **RecordMap** maps a single **FixedRecord** to a single **DelimitedRecord**.

To create a functional map

1. In the Navigator, select the map source file in which to create the desired functional map.
2. From the **Map** menu, choose **New**.
The Create New Map dialog box is displayed.
3. Enter the name of the map to be created in the **New map name** field.
4. Click **OK**.

The new map is created and appears in the Navigator. In **List** view, the functional map is at the same level as the executable map. In **Composition** view, the functional map appears at a level beneath the executable map that calls it.

To create input and output cards in the functional map

1. Create the required input cards. Each input card represents a single data object. The **TypeName** of the input card in a functional map must coordinate with the input argument(s) in the map rule on the executable map. If there is more than one input card in a functional map, ensure that the input card types correspond to the order of the input arguments.
2. Create the required output card. Each output card represents a single data object.

- Functional maps have only one output card.
3. Enter map rules to create the output of the functional map.
 4. Save the functional map.
 5. Build the executable map.
 6. Save and run the executable map.

Input card types of a functional map

The input and output cards in a functional map must be in the same order as the data objects are specified in the map rule argument.

The input card type(s) in a functional map must be coordinated with the input argument(s).

For example, if the input arguments evaluate to **Record**, **Claim**, and **LookupFile**, in that order, then the input card types in the functional map must be **Record**, **Claim**, and **LookupFile**, in the same order.

Input arguments as data objects

Input arguments of a functional map can evaluate to **Item** or **Group** type data objects.

Item types

When an input argument of a functional map evaluates to an item type, the type of the corresponding input card must be an item type with the same **Item Subclass** (**Number**, **Text**, **Date & Time**, or **Syntax**).

An input argument might evaluate to a number. If the following input argument evaluates to a number, the corresponding input card type must be an item with an **Item Subclass** of **Number**:

INDEX (Record:File)

If the input argument is a text literal (for example, "payroll"), the input card type that corresponds to this argument must be an item with an **Item Subclass** of **Text**.

Group types

When an input argument of a functional map evaluates to a group type, the type of the corresponding input card must be the exact same group. For example, if the second input argument evaluates to **InvoiceSet**, the **TypeName** of input card #2 must be **InvoiceSet**.

If there is one argument of the functional map, there will be one input card. The **TypeName** on the input card must correspond to the data object specified in the argument for the functional map.

Functional maps do not require **InputData** and **OutputData** settings.

When **InputData** and **OutputData** settings exist in input and output cards in a functional map, they are ignored when the executable map is compiled.

Output card type of a functional map

The type of the output card in a functional map is the type whose map rule includes the functional map name. For example, if the functional map name is entered in the rule cell for **CarRecord(s)**, the output of that functional map must be **CarRecord**.

The output of a functional map must match the output data object (where the map rule that calls the functional map is), therefore a functional map has only *one* output card.

A functional map may have only *one* output card.

Entering map rules

Map rules of a functional map are entered to generate to the desired output of the functional map. Similar to an executable map, output data is generated in the specified format. This specified format is determined by the type properties of the type on the output card.

Essentially, a function maps a single data object for every data object received from the input.

Using multiple inputs in a functional map

A functional map may require more than one input. For example, you are mapping an input purchase order to an output purchase order. The input **PO** consists of a header and multiple details. The output **PO** is composed of a series of item records. Each **ItemRecord** contains the **PO#** and the company name, which must be mapped from the **Header** of the input **PO**. Each **ItemRecord** also contains item, quantity and price information, which is mapped from a **Detail** in the input.

You need a functional map to generate an **ItemRecord**. The inputs of the functional map would be everything that is necessary to generate one **ItemRecord**: both the **Detail** and **Header** from the input. A suitable name for the functional map may be **MakeItemRecord**.

The input **PO** is composed of one **Header** and multiple **Details**. You want to generate one **ItemRecord** for each **Detail**.

The functional map **MakeItemRecord** takes one **Detail** at a time, along with the **Header**, and creates one **ItemRecord**.

The two input arguments of **MakeItemRecord** are **Detail** and **Header**. You would enter the following map rule for the output **ItemRecord(s)**:

```
= MakeItemRecord (Detail:Input, Header:Input)
```

In the map **MakeItemRecord**, you would create two input cards. The first input card corresponds to the first input argument, **Detail**. The second input card corresponds to the second input argument, **Header**.

- The input arguments must match the input card types, in the same order.
- The order of the arguments does not affect the number of times a rule is evaluated. However, it may affect the order in which the output objects appear and the performance at runtime. For more information on how rules are evaluated, see the Functions and Expressions documentation.

In **MakeItemRecord**, map the **PO#** and **Company** from the **Header** into the components of **ItemRecord**. Map the **Item**, **Qty**, and **Price** from the **Detail**.

Input objects are triggers

A functional map is triggered by the presence of the input objects that are specified as the arguments of the functional map. Each time a new combination of input objects is evaluated, the functional map is executed.

The functional map **MakeItemRecord** has two inputs: **Detail** and **Header**. One output **ItemRecord** is produced for each combination of **Detail** and **Header** in the input. In this example, there is one **Header** for many **Details**. The same **Header** object is sent to the functional map again and again with each new **Detail**.

For a complete explanation of how map rules are evaluated, see the Functions and Expressions documentation.

Going to a functional map referenced in a map rule

You can easily go to a functional map whose name is in a map rule.

To open and view the functional map referenced in a rule:

1. Right-click the rule containing the name of the functional map.
2. From the context menu, choose **Goto Functional Map**.
3. Select the functional map to which you want to go.
4. Click **OK**.

Functional maps in the Navigator

The composition of maps is displayed in the Composition view of the Navigator. A map that is used as a functional map is displayed underneath the map that calls it.

Functional maps that do not exist

If a map rule includes a map name that does not exist in that map source file, the map name appears a pseudo map in the Navigator.

Dragging a functional map name into a rule

You can enter the name of a functional map by dragging it from the Navigator into the rule bar.

To enter the name of a functional map into a rule:

1. Select the rule where you want to enter the functional map name, making sure it has current focus.
2. Drag the functional map name from the Navigator into the rule bar.

Chapter 10. Functional Map Wizard

Use the Functional Map Wizard to create a functional map (or maps) based on a map rule (or rules). The maps that the Wizard creates contain input cards and an output card, but do not contain map rules.

Using the Functional Map Wizard

The functional map creation requires a map rule on an output card of an executable map.

To use the Functional Map Wizard:

1. In the executable map, enter the map rules for the functional map.
2. Select the rule, or rules, for which you want to generate a functional map, or maps.
3. From the **Rules** menu, choose **Functional Map Wizard**.
The Functional Map Wizard dialog box is displayed.
4. If necessary, correct problems related to undefined cards and maps.
5. Edit the cards as required. It is recommended to assign meaningful names to the cards in the functional map. A good tip is to name the card the same name as the data object it represents.

After a map is successfully created, an icon appears next to the map name and the cards for that map.

6. Select a map to create and click **Create**.
The map is created in the active map source file. You can then modify the functional map and enter the appropriate map rules.
7. Continue doing this until all the maps you want to create with the wizard are created.
8. Click **Close** to close the wizard dialog box.

Properties of the functional map

The Functional Map Wizard creates maps based on the map rules selected when the wizard is activated.

Map name

The name of the functional map to be created is specified in the map rule. The name of the functional map must be unique within the map source file.

To easily identify functional maps, you may use a specific naming convention. Using the **F_** prefix for all functional map names is a common practice. The functional map **F_RecordMap** is easily identified.

Input cards

An input card is created for each functional map argument specified.

Input card types

When an argument is an object name, the type for the corresponding input card is known.

When an argument is a function that returns an input object, the type for the corresponding input card is known. It is the type of that input object returned from the argument. However, when an argument is a literal, or an expression whose output does not correspond to an input object name, an icon appears next to that undefined card. You must edit the input card and select the type.

For example, an argument is `COUNT (Set:Input)` (which returns a number) and the card is undefined. To define the card, you must choose the type that the card represents.

For information on choosing a type within the Functional Map Wizard dialog box, see "When an Argument Returns an Object of a Known Type" .

Input card names

The Functional Map Wizard creates cards with generic names. Edit the card names to reflect that data object it represents.

The default name of each input card created by the Functional Map Wizard is **In** and the number of the card. For example, the second input card has the name **In2**.

Output card

The Functional Map Wizard creates one output card. There is only one output card in a functional map.

Output card type

When specifying the type of the output card in the functional map, the type is where the rule is located. For example, if the rule you select is on the output **DelimitedRecord(s)**, the type of the output card in the functional map will be **DelimitedRecord**.

Output card name

The default name of the output card created by the Functional Map Wizard is **Out**. You can edit this name in the Functional Map Wizard dialog box.

Referencing a map from the Functional Map Wizard

From **Functional Map Wizard**, you can create a new functional map by reference. For example, if you use the same functional map across several map source files, instead of creating that specific functional map for each map source file, you can create the functional map within one map source file and then use it in other map source files simply by creating references to it.

When using the Functional Map Wizard, use the following procedure to create a functional map by reference.

To reference a map from the Functional Map Wizard:

1. Select the rule in your output card and click **Rules** → **Functional Map Wizard**.
The Functional Map Wizard dialog box is displayed.
2. Click **Reference**.
The Map Reference Selection dialog box is displayed.

3. In the **Referenced Map Source File** field, enter the map source file that contains the map you want to reference.
The map contents appear under **Contents**.
4. Select a map and move it under **Selected Files**.
5. Click **OK** to return to the Functional Map Wizard.
At this point the referenced functional map is created, however, the appearance of the Functional Map Wizard dialog does not change.
6. Click **Close**.
The referenced functional map is created and appears in the Navigator under the appropriate map source file.

Editing card definitions in the Functional Map Wizard

You can edit card definitions in the Functional Map Wizard dialog box. It is a good idea to assign meaningful names to the cards.

It is not a good idea to change the type of a card that the wizard has created. After all, the wizard chooses the correct types, based on the map rule. However, there are cases in which it is necessary to choose the type of an undefined card.

To edit card definitions in the Functional Map Wizard:

1. Select the card in the Functional Map Wizard dialog box and click **Edit**.
The Card Attributes dialog box is displayed.
2. Make changes.
3. Click **OK**.

Resetting the Functional Map Wizard

If you edit any settings in the Functional Map Wizard for a card in the Functional Map Wizard dialog box, you can return to the original settings.

To reset the Functional Map Wizard:

1. Select the map or card to be reset.
2. Click the **Reset** button.
The **Card Name**, **Tree**, and **Name** settings are restored.

Undefined card and maps

If an input type is undefined, you must supply the necessary information.

To determine missing or undefined information:

1. Select the card that is undefined.
2. Click the **Info** button.
The information on the selected card and type is provided.

When an argument is invalid

If an argument of a functional map in a rule is invalid, no card is created for that argument. You must fix the map rule and try the wizard again.

For example, the following invalid rule for **DelimitedRecord(s)** is entered:

```
=RecordMap (FixedRecord)
```

The object name for **FixedRecord** is invalid because **FixedRecord** is a component of **Input**. The rule should be:

```
= RecordMap (FixedRecord:Input)
```

The type cannot be determined for the input card, because the object name in the map rule is incorrect. When invalid arguments exist, close the Functional Map Wizard dialog box and fix the map rule.

Select the invalid card, and click the **Info** button to view the message. The invalid argument message is corrected by editing the map rule that calls the functional map.

When an argument returns an object of a known type

Sometimes an argument of a map is not simply an object name; it is a more complex expression. The Functional Map Wizard creates maps that have complex arguments only if the type of the object the argument returns can be determined.

For example, in the following map rule:

```
= F_ResponseMap (EXTRACT (Customer  
Record:ResponseList::ActivityReport,  
Call Field:Customer Record:ResponseList::ActivityReport =  
"Y"))
```

The first argument of **ResponseMap** is an **EXTRACT** expression that, in this case, returns the type **Customer Record**. The Functional Map Wizard creates the **F_ResponseMap** functional map, and chooses the correct type for the input, **Customer Record**.

Creating more than one functional map

The Functional Map Wizard can create all of the functional maps in a map rule. For example, if a map rule contains the name of two maps, the wizard creates both maps.

The following map rule on the output object **OutputPO(s)** refers to two maps:

```
= IF (PO#:InputFile < 1000, F_OldPO (InputPO:InputFile),  
F_NewPO(InputPO:InputFile))
```

The rule refers to two maps: **F_OldPO** and **F_NewPO**. The Functional Map Wizard creates both the **F_OldPO** and the **F_NewPO**.

Selecting multiple map rules

You can select multiple map rules and run the Functional Map Wizard. For example, select the following two map rules to create two functional maps:

```

= F_EventMap (Activity:ActivityReport)
= F_ResponseMap (EXTRACT (Customer
Record:ResponseList::ActivityReport,
Call Field:Customer Record:ResponseList::ActivityReport =
"Y"))

```

The maps for all of the rules you select are created.

If a map already exists

If the map names in the rules you specify in the map rule to generate a functional map already exist, the maps are not created and a message indicating that there are no unknown maps appears.

To create the specified map when a map with that name already exists, rename the map in the rule, or rename the existing map.

Using an output as an argument

You can use an output data object as an argument of a functional map. The object being used as an argument of a functional map must be used on an earlier output card in the same map source file.

To specify an output data object as an argument of a functional map:

1. Select the output card containing the data object.
2. Float the To window to view two cards at the same time. Press Ctrl and drag the To window to the main window.
3. In the map rule on the second output card, enter the name of the functional map.
4. Drag the data object from the first output card to the rule bar to use it as an argument for the functional map.

Multiple levels of functional maps

Functional maps can have multiple levels of subsequent functional maps.

For example, you can use two levels of functional maps with an input of multiple purchase orders that must output multiple invoices. In this case, there is a one-to-one relationship between an input purchase order (PO) and an output invoice, which indicates the need for a functional map that can generate one output invoice for each input PO.

- The **BatchMap** executable map maps the entire input of multiple POs to the output set of invoices.
- **BatchMap** calls the **POToInvoice** map, which maps each PO to one invoice. The **POToInvoice** map can be used as a functional map.
- **POToInvoice** calls the functional map **ItemMap**.
- The **ItemMap** functional map maps each specified item of data. The type trees used in **POToInvoice** must be the same trees used in **BatchMap**.

In **BatchMap**, the map rule on the **Invoice(s)** output data object references the map **POToInvoice**. The input argument of **POToInvoice** is **PO:POBatch**. This ensures that one invoice is generated for each PO.

Although this example only uses two levels, the number of levels of functional maps you can use within a functional map is unlimited.

Using a functional map to increment data

A functional map can only map input or output data that is passed to it as an argument. Therefore, to generate consecutive line item numbers for multiple occurrences, use the INDEX function on objects as an input argument of the functional map. Each time that data object is called, the index of the current object will be sent to it.

For example, when **ItemMap** is called for the first time, the index of the **LineItem** will be 1. When **ItemMap** is called for the second time, the index of the **LineItem** will be 2. This number will then be mapped to the field **Item#**.

Referring to the current output in a map rule

A map rule can include the name of the output object to which it applies. For example, to use the INDEX function on **LineItem**, the map rule on **LineItem** will include **LineItem**. Instead of using the entire object name for **LineItem**, use the dollar sign shorthand symbol \$. The dollar sign \$ refers to the object to which the expression applies.

A map rule can also reference other output objects that appear above the map rule.

In the functional map, you can create an additional input card to represent the index of the desired item. This is one case in which the type of an input card in a functional map has been chosen from the output. In fact, *any* numeric item may be chosen for that card's type.

Using a functional map to reorder data

Use a functional map to arrange data in a different order. For example, you are growing some flowers and tracking their progress. You have some data for the measurements you took: height, number of leaves, water intake, and so on. These **ProgressReadings** are in no particular order in your input data. For example, the readings for daisy are scattered throughout the input data.

Shown below is some input data:

Apr-06,Alstromeria,24.14,15,7,32.5

Mar-20,Daisy,15.3,13,12,3.67

May-02,Lily,10.4,7,13,2.5

Apr-22,Daisy,22.4,8,12,2.6

Mar-09,Snapdragon,10.2,7,10,3.2

May-11,Alstromeria,26.8,18,11,3.12

Apr-06,Iris,5.89,5,6,2.7

Apr-10,Daisy,18.4,8,13,4.1

You want to generate an output that consists of reports, one for each flower type. You want a report for alstromeria, one for daisies, one for snapdragons, and so on. Use a functional map called **MakeReport**, to produce one report per flower type.

To create a report for each flower type, you need to determine the different kinds of flowers in the input data. To do this, use the **UNIQUE** function. The **UNIQUE** function looks at all of the values for a given data object and returns the unique values.

In the input data, the object called **Flower** indicates the flower type. Use the **UNIQUE** function on **Flower** as the first input argument to the map **F_MakeReport**. This will cause the map **F_MakeReport** to be executed once per flower type.

In the input data, the unique flowers are alstromeria, daisy, snapdragon, lily, and iris. In each output report for a given flower type, you want all of the **ProgressReadings** for that flower type, which are scattered throughout the entire input data. To get all of these **ProgressReadings**, you will use the entire input object as the second argument to the map **F_MakeReport**.

Chapter 11. Map settings

Map settings specify properties for map executions that are not specific to a particular input or output. These include settings for auditing a map, tracing data content, performance settings, validation settings, and map-specific warnings and errors.

Map settings are defined for a map and compiled with a map.

- In the Map Designer, the map settings defined in the Tools > Options > Run Options > MapSettings dialog box are the default settings that apply when a new map is created.
- Map settings are defined for a map at design time in the Map Settings dialog box and are compiled with that map.
- The map settings compiled with that map can be overwritten at run time in the Command Server, Launcher, Integration Flow Designer, and in adapter commands.

Defining map settings for a single map

When a map is run using a command line, overrides for all map settings except **BurstRestart** are also available.

To change map settings for a single map

1. Select the map in the Navigator.
2. From the **Map** menu, choose **Settings**.

The Map Settings dialog box is displayed with the name of the active map in the title bar.

3. Modify the map settings to be compiled in the active map.
4. Click **OK**.

Defining default map settings

Map settings are compiled with a map. In the Map Designer, the map settings defined in the Map Settings dialog box are the default settings that apply when a map is created.

Each new map created uses the default map settings specified in the **Run Options**.

To define default map settings:

1. From the **Tools** menu, choose **Options**.
2. In the Options dialog, select **Run Options**.

The **MapSettings**, **DisplaySettings**, and **Configuration File** are displayed in the right pane of the Options dialog box.

3. For Setting, define the desired default.
4. Click **OK**.

MapAudit settings

MapAudit settings specify options for creating and storing map audit logs.

If the **MapAudit Switch** is set to **ON**, the audit log is created and can be appended. In addition, there are flexible sub-switches that allow for the audit log to be created or appended based on the following conditions:

- When a map fails.
- If the map has an error.
- If the map has a warning or error.

MapAudit > Switch

The **Switch** setting enables or disables the map audit feature. The default value for the **Switch** setting is **OFF**.

Value	Description
-------	-------------

OFF	Audit data for the map will not be created.
-----	---

ON	Audit data for the map will be created based on the values of the related settings.
----	---

MapAudit > BurstAudit

Expand this setting to define the audit log **BurstAudit Data** and **BurstAudit Execution** settings.

BurstAudit > Data

The **Data** setting is used to include or exclude data information in the audit log. The burst audit data information includes a burst count, the card number, the object index, the audit log status code, the object level, the size of the object, and the object name. You can specify that you want to audit a particular object by including that object in the Organizer window on the **Data Audit Settings** tab.

The **BurstAudit Data** settings are listed in the following table. The default setting is **Never**.

Value	Description
-------	-------------

Always	Write data to the audit log, regardless of the map execution outcome.
---------------	---

Never	Do not log data information.
--------------	------------------------------

OnBurstError	If the outcome of the execution of the burst meets a failure condition, write the burst data to the audit log.
---------------------	--

OnBurstWarningOrError	If the outcome of the execution of the burst is a warning or meets a failure condition, write the burst data to the audit log.
------------------------------	--

Data > SizeValidation

When **BurstAudit Data** is set to **Always**, **OnBurstError**, or **OnBurstWarningOrError**, the **SizeValidation** field becomes available. The input trace file will report when data fails to meet size requirements for the type.

Value Description

TooLong TooShort

When you choose this option and the minimum or maximum size requirements are not met during data validation, one of the following size-specific messages will be indicated in the input trace file:

"fails minimum size requirements"

"fails maximum size requirements"

Wrong Size

This is the default option for this setting. If the minimum or maximum size requirements are not met during data validation, it will be indicated in the input trace file with a generic message similar to the following example:

"data at offset xx ('DATA...')is the wrong size for the TYPE X'(02)
(typename)"

When using the **TooLong TooShort** value for the **BurstAudit Data SizeValidation** setting and the data fails the minimum size requirement, the input trace file entry will look similar to the following example:

```
(Level 1: Offset 41, len 6, comp 5 of 7, #1, DI 00000005:)  
Data at offset 41 ('SYNTAX...') fails minimum size requirements  
TYPE X'0007' (LongSyntaxItem ROOT).
```

BurstAudit > Execution

The **Execution** value determines whether to include audit execution information in the audit log.

The execution audit log includes the command used to run the map from a command line, the number of input and output objects, and information about data sources, data targets, work area size, and the time it took to run the map. The execution log does not include information about restarts.

The **BurstAudit Execution** settings are listed in the following table. The default setting is **Never**.

Value Description

Always

Include burst audit execution information in the audit log, regardless of map execution outcome.

Never Do not include burst audit execution information in the audit log, regardless of map execution outcome.

OnBurstError

If the burst execution outcome meets a failure condition, write burst audit execution information to the audit log.

OnBurstWarningOrError

If the burst execution outcome is a warning or meets a failure condition, write burst audit execution information to the audit log.

MapAudit > SummaryAudit

Expand this setting to define the summary audit **Execution** settings. Summary audit provides audit information on map execution without the detailed burst information.

SummaryAudit > Execution

The **Execution** value determines whether to include summary audit execution information in the audit log. Summary audit provides audit information on map execution without the detailed burst information.

The **SummaryAudit Execution** settings are listed in the following table. The default setting is **Always**.

Value	Description
-------	-------------

Always	
---------------	--

	Include summary burst audit execution information in the audit log, regardless of the map execution outcome.
--	--

Never	Do not include summary burst audit execution information in the audit log, regardless of the map execution outcome.
--------------	---

OnError	
----------------	--

	If the map execution outcome meets a failure condition, write the execution information to the audit log.
--	---

OnWarningOrError	
-------------------------	--

	If the map execution outcome is a warning or meets a failure condition, write the execution information to the audit log.
--	---

SettingsAudit

Expand this setting to define which settings are included in the audit log.

SettingsAudit > Data

The **Data** value determines whether to include data settings in the audit log.

The **SettingsAudit Data** settings are listed in the following table. The default setting is **Never**.

Value	Description
-------	-------------

Always	
---------------	--

	Include data settings in the audit log, regardless of the map execution outcome.
--	--

Never	Do not include data settings in the audit log, regardless of the map execution outcome.
--------------	---

OnError	
----------------	--

	If the map execution outcome meets a failure condition, include data settings in the audit log.
--	---

OnWarningOrError	
-------------------------	--

	If the map execution outcome is a warning or meets a failure condition, include data settings in the audit log.
--	---

SettingsAudit > Map

The **Map** setting is used to include or exclude the settings from the map in the audit log.

The **SettingsAudit Map** settings are listed in the following table. The default setting is **Never**.

Value	Description
-------	-------------

Always	Include map settings in the audit log, regardless of map execution outcome.
---------------	---

Never	Do not include map settings in the audit log, regardless of map execution outcome.
--------------	--

OnError	If the map execution outcome meets a failure condition, write map setting information to the audit log.
----------------	---

OnWarningOrError	If map execution outcome is a warning or meets a failure condition, write map setting information to the audit log.
-------------------------	---

MapAudit > AuditLocation

The **AuditLocation** setting is used to specify where the audit information is stored.

The availability of additional settings (such as **Directory**, **FileName Action** **LogFileAction** or **Sized**) varies according to the **Location** setting.

The **MapAudit AuditLocation** values are listed in the following table. The default value is **File**.

Value	Description
-------	-------------

File	Audit information is stored in a file. There is a maximum of 259 characters in the path and filename.
-------------	---

Memory	Audit information is stored in memory, and appended to the last set of data returned during map execution. The Memory value is used when the map is run by the RUN function or by an API.
---------------	--

When a map is run by a Command Server or Launcher, the **Memory** setting is ignored. The audit log is created in a file in the map directory, as specified with the **AuditLocation FileName** setting.

Directory

The **Directory** setting is used to specify the same directory as the map or a user-defined directory. The default value for the **Directory** setting is **Map**.

Value	Description
-------	-------------

Map	The same directory as the map.
------------	--------------------------------

Custom	The directory you specify using the Value setting.
---------------	---

FileName

Use the **FileName** setting to specify a default or custom name for the audit log file.

Value Description

Default

The file is automatically assigned an audit log file name based on the compiled map name with the file name extension **.log**.

Unique

Generates a unique audit log name in the following format:

Mer_*<mapname>*_*<process_key>*_*<map_counter>*.**log**

Run_*<mapname>*_*<process_key>*_*<map_counter>*.**log**

The prefix "Mer" is used with top-level maps while "Run" is used for run maps.

- *<mapname>* is the executable map name.
- *<process_key>* is a unique, per process value.
- *<map_counter>* is a unique, per map instance value (within the same process).

If you choose a **Unique** filename for the audit log file, this file cannot be viewed from the Organizer window.

Custom

Specify the custom name of the audit log file using the **FileName > Value** setting.

Use the **Custom** value if the log files produced for one map trigger another map, so that each log file will trigger another instance of that other map (be sure to specify different directories for the log files).

The amount of characters allowed for the log filename is contingent on the number of characters in the directory path. The total number of characters in the path name and the file name is limited to a maximum of 255 characters.

Value

When the location setting is **Custom**, enter the full path of the directory in the Value setting or browse to select the directory.

FileName > Action

The **Action** setting is used to create a new audit log file or to append the audit log to an existing audit log file. The default value for the **LogFileAction** setting is **Create**.

Value Description

Create The audit log file is created each time the map is run.

Append

The audit log is appended to an existing audit log file each time the map is run. If no audit log file exists at the start of map execution, an audit log file is created.

FileName > Value

Specify a custom filename. The amount of characters allowed for the audit log filename is contingent on the amount of characters in the directory path. The total number of characters in the path name and the file name is limited to a maximum of 255 characters.

AuditLocation > Sized

When **AuditLocation = Memory**, audit information is stored in memory and appended to the last set of data returned during map execution. The memory setting can be used only when the map is run by the RUN function or an API. Audit data is prefixed with a text number indicating the size of the audit data, followed by a <SP>.

Value	Description
-------	-------------

ON	The audit data is sized.
----	--------------------------

OFF	Not sized: the audit data is returned with no size.
-----	---

MapTrace settings

MapTrace settings specify properties for a trace file used to diagnose invalid data or incorrect type definitions.

ContentTrace

Specifies that the trace information is written to a file.

ContentTrace > Switch

Define whether map tracing is enabled. A trace file can be used for diagnosing invalid data or incorrect type definitions.

Value	Description
-------	-------------

OFF	Trace data for the map will not be created.
-----	---

ON	Trace data for the map will be created.
----	---

ContentTrace > TraceLocation

Use the **Location** settings to specify where the trace file will reside. The **TraceLocation** is **File**, which specifies that trace information is written to a file as specified with the **Directory** and **FileName** settings.

Directory

The **Directory** setting is used to specify the same directory as the map or a user-defined directory. The default value for the **Directory** setting is **Map**.

Value	Description
-------	-------------

Map	The same directory as the map.
------------	--------------------------------

Custom	
---------------	--

The directory you specify using the **Value** setting.

Value

When the **Directory** setting is **Custom**, enter the full path of the directory in the **Value** setting or browse to select the directory.

FileName

Use the **FileName** setting to specify a default or custom name for the trace file.

Value	Description
-------	-------------

Default	The file is automatically assigned the <i>mapname</i> with the file name extension .mtr. (For example, ContactToLabel.mtr)
----------------	--

Custom	Specify the custom name of the trace file using the FileName > Value setting.
---------------	---

Value

When the location setting is **Custom**, enter the full path of the directory in the **Value** setting or browse to select the directory.

InputContentTrace

Specify options of which input data is traced. The default value for the **Detail** setting is **All**.

Value	Description
-------	-------------

Off	Trace input data for the map will not be created
------------	--

All	Trace all inputs.
------------	-------------------

Card#	Trace a single input. Specify the input map card by typing the number of the map card into the value field of the Card# setting.
--------------	---

Range	Trace a range of input objects. Specify the beginning object by typing the number of the object into the value field of the Start setting. Specify the ending object by typing the number of the object into the value field of the End setting.
--------------	--

CardNumber

To trace a single input trace, enter the card number. For example, to include only messages about card 2 in the trace file, enter 2.

InputContentTrace > StartObject

Enter the beginning object number to be traced. For example, to trace data objects 1000 to 2000, enter 1000 as the **Start** value.

InputContentTrace > EndObject

Enter the ending object number to be traced. For example, to trace data objects 1000 to 2000, enter 2000 as the **End** value.

Using ranges for InputContentTrace

The range specified for objects is not precise. The intention is to reduce the number of objects in the trace file, not to specify precise objects. To trace the first object, specify a **Start** range of 0 (not 1).

The object number is not provided in the trace file. Providing a range is an attempt to trace less than the whole, not a precise range of objects.

RulesTrace

The **Detail** setting specifies options of which output data is traced. The default value for the **Detail** setting is **OFF**.

Value Description

OFF Trace output data for the map will not be created.

All Trace all objects built.

Card Trace a single output. Specify the output map card by typing the number of the map card into the value field of the **Card#** setting.

CardNumber

To trace a single output trace, enter the card number. For example, to include only messages about card 2 in the trace file, enter 2.

SummaryContentTrace

The **SummaryTrace** setting is used to place a summary of the validation status of each input and the build status of each output after the map is built into the trace file. The default value for the **SummaryTrace** setting is **OFF**.

Value Description

OFF Do not include a summary in the trace file.

ON Include a summary in the trace file.

Workspace settings

Workspace settings specify the location (file or memory) of the workspace and **PageSize** and **PageCount** settings for configuring the page size. **Workspace** settings allow you to optimize the speed of map processing.

When a map is run, a paging scheme to access information about the data as efficiently as possible is used. Portions of both working data and actual data are paged, as needed. The workspace can be a combination of files and memory, or simply memory-based. When file-based workspace is used, the amount of memory accessed at a given time depends on the number of pages and the count of pages configured for a map to run.

The **Workspace** settings specify the location (file or memory) of the workspace and **PageSize** and **PageCount** settings for configuring the page size.

The default value for the **Location** setting is **File**.

Value Description

File A work file is created for each input card. Use the **Directory** setting to specify the directory.

Memory

A memory-based workspace is created.

The **WorkSpace** settings allow you to optimize the speed of map processing.

- When the size of the input data a map is processing is large, file-based workspace using default paging is recommended (**Workspace = File**).
- When small amounts of data are processed, smaller page sizes and memory-based workspace is recommended (**Workspace = Memory**).
- When using burst processing, adjust the workspace configuration for the size of burst data.

If a particular map card is not referenced by any map rules, no work area for that card will be created, and the audit log will indicate that the work area is not found.

PageSize

The **PageSize** setting specifies the size of a memory page to be used for work files. For maps that include a large number of type references, you may want to increase the page size. The default value for the **PageSize** setting is **64**.

Value Description

n Enter a number that represents the size of a memory page in kilobytes. The page size should be a multiple of 2, between 2 and 1024.

PageCount

The **PageCount** setting specifies the number of memory pages to use for work files. For large data files, you may want to increase the number of pages used. The default value for the **PageCount** setting is **8**.

Value Description

n Enter a number that represents the number of memory pages to use for workspace. Valid numbers are 1 to 9999 (an integer greater than 0).

Directory

The **Directory** setting is used to specify the same directory as the map or a user-defined directory. The default value for the **Directory** setting is **Map**.

Value Description

Map The same directory as the map.

Custom

The directory you specify using the **Value** setting.

WorkFilePrefix

The **WorkFilePrefix** setting is used to select a convention for naming workfiles. The default value for the **WorkFilePrefix** setting is **MapName**.

Value Description

MapName

The name of a work file is the base name of the compiled map file. The extension is *.Inn*, for an input, and *.Onn* for an output, where *nn* is the card number. For example, the work file for input card number 3 would have the extension *.I03*.

Unique

Generates unique work files with no conflict between work files each time a given map is run. If this option is turned on, work files are automatically deleted.

WorkFilePrefix > Action

Use the **WorkFileAction** setting to save or delete the work files. The default value for the **WorkFileAction** setting is **Delete**.

Value Description

Delete Delete the work files after a map is executed. All work files are deleted, except those being saved for reuse, as specified on the input cards for each map.

Create Save the work files based on the **Value** and **FilePrefix** settings.

Century settings

Century settings specify the current or sliding century.

Century

The **Century** setting specifies the current or sliding century. The default value for the **Switch** setting is **Current**.

Value Description

Current

Missing century numbers are derived based on the current system date.

Sliding

Enables **SlidingCentury** options. Missing century numbers are derived based on the value of the **CCLookup** setting.

Century > CCLookup

The **CCLookup** setting specifies the first year in a 100-year range used to derive a missing century for a date item, using CCYY format. The default value for the **CCLookup** setting is **1950**.

For example, if the value of the **CCLookup** setting is **1960**, a YYYYMMDD formatted date between 1960 and 1999 is interpreted as having a century date of **19**, while dates between 2000 and 2059 are interpreted as having a century date of **20**. In this case,

- 600101 is assigned a century of 19
- 580101 is assigned a century of 20

Value Description

n Enter a number in CCYY format that represents the start of a 100-year range. Valid numbers are 1 to 9999.

Validation settings

Validation settings specify how strictly the data is validated by specifying response behavior for types of validation errors.

Validation

During map execution, the input data is validated to assure it matches the definition of that input. Specify how strictly the data is validated by specifying response behavior for types of validation errors.

Value	Description
-------	-------------

Standard	
-----------------	--

	Standard validation occurs when the map is run.
--	---

Custom	
---------------	--

	Enables custom validation options. Define these options in the OnValidationError , RestrictionError , SizeError , and PresentationError settings.
--	---

OnValidationError

Under certain circumstances, multiple errors are detected that occur during validation. The **OnValidationError** setting defines where to halt map execution. With either setting, if an error is found in the input, no output objects are built.

Value	Description
-------	-------------

Stop	Only validate through the input card where the first error occurs. Stop executing the map after validating that input card, even if there are multiple input cards.
-------------	---

Continue	
-----------------	--

	Continue validating data until all errors are found. Even when validation finds data errors in a given input card, map execution continues with the next input card.
--	--

RestrictionError

Define whether to validate the data with restrictions at runtime. For example, if you are running some test data and you do not need the item values to match the restrictions, you might wish to ignore the restrictions.

Value	Description
-------	-------------

Ignore	Ignore restrictions during validation.
---------------	--

Validate	
-----------------	--

	Validate with restrictions.
--	-----------------------------

Ignore (no warnings)	
-----------------------------	--

	Allows restrictions to be fully ignored, in all situations other than when ignoring is required for object identification, such as when an item is partitioned by a restriction list.
--	---

Restrictions of syntax items are not ignored, because these are used in determining the value of the syntax object.

SizeError

Define whether to validate the data with size at run-time.

Value	Description
-------	-------------

Ignore	Ignore minimum and maximum size of items that appear in delimited data during validation.
---------------	---

Validate	Validate with size.
-----------------	---------------------

For example, if an item is defined as having a minimum of 3 bytes, but it only has 2 bytes in the data, the fact that it only has 2 bytes is ignored and the data is validated anyway.

PresentationError

Define whether to validate the data with presentation settings at run-time. For example, if a number is defined as an integer, but it appears in the data as a decimal number, the fact that it appears as a decimal is ignored and the data is validated anyway.

Value	Description
-------	-------------

Ignore	Ignore presentation settings during validation. If the item has an invalid presentation, the item is mapped as if it were NONE. However, if the input is converted to text using the TEXT function, it <i>will</i> be mapped.
---------------	--

Validate	Validate with presentation settings.
-----------------	--------------------------------------

Retry settings

The **Retry** map settings define whether to try again if compiled map files, work files, audit log files, or trace files that are needed during map execution cannot be opened.

Specify the **Interval** and **MaxAttempts** to define how many times and at what interval attempts are made to access an unavailable file.

If adapter errors occur (for example, failure to connect) and **Retry Switch = On**, the adapter process is retried at the interval specified with the **Interval** value up to as many times as defined with the **MaxAttempts** setting.

Retry > Switch

The **Switch** setting enables or disables **Retry** options. The default value for the **Switch** setting is **OFF**.

Value	Description
-------	-------------

OFF	Disables Retry options. Only one attempt to access an unavailable file. If this fails, a return code and message (Could not open file) are returned by the map.
------------	--

ON	Enables Retry options. Retry is enabled for the number of attempts specified by the value of the MaxAttempts setting at the interval specified by the value of the Interval setting.
-----------	---

Retry > Interval

Use the **Interval** setting to specify the interval to wait between attempts to open a system file referenced by the map. The default value for the setting is 0.

Value	Description
-------	-------------

n	Enter a number that represents the number of seconds to wait between attempts to access input data referenced by the map. Valid entries are integers 0 to 9999.
---	---

Retry > MaxAttempts

Use the **MaxAttempts** setting to specify the maximum number of attempts to access an unavailable file referenced by the map. The default value is 10.

Value	Description
-------	-------------

n	Enter a number that represents the maximum number of attempts to access an unavailable file referenced by the map. After the maximum number of attempts has been exhausted unsuccessfully, a return code and message are returned by the map. Valid entries are integers 0 to 65535.
---	--

Warnings settings

Warnings settings specify that the map should fail when any of the selected warning codes are returned. The default value for the **Warnings** setting is **Every**.

Enabling the **Warnings** options impacts the transaction behavior specified for each card in the map.

If a map execution warning code occurs, the map fails after processing of that map is finished.

Maps called (from the failed map) with a RUN function, PUT function, or GET function will still process, and the behavior defined in that map's card settings for **Transaction OnSuccess** and **Transaction OnFailure** occurs.

If any warning is set to **Fail** the map, and the map fails due to the cause associated with the selected warning, the execution audit includes the map-failed message with the appropriate warning code preceded by a minus sign, for example: Map failed on warning (-27).

Define your audit settings for each map, as the audit log of the main map may not reflect all failed map information if the failed maps are called with a RUN, PUT, or GET function.

Value	Description
-------	-------------

Every	Map fails when any of the warning codes are returned. This setting treats every warning the same way, depending on the value of the Return setting. For Return , you can select Warn , Ignore , or Fail .
--------------	--

Custom	Map fails only when the user-selected warning codes are returned. The specific warnings are available only if the value of the Warnings setting is Custom .
---------------	---

Warnings > Return

The **Return** setting specifies the response for warnings. The default value for the *Return* setting is **Warn**.

Value	Description
--------------	--------------------

Warn	Issue a warning if any warning is returned.
-------------	---

Ignore	Ignore all warnings if any warning is returned.
---------------	---

Fail	Fail the map if any warning is returned.
-------------	--

Chapter 12. Configuring bursts

For input card settings that have a **FetchAs** mode of **Burst**, the type of input object (as defined in the **Source** setting) and the **FetchUnit** work together to achieve the desired results.

For example, if an input type is one message, the **FetchUnit** should be 1. If the input type is a series of messages, the **FetchUnit** can be any integer.

- When the **Source** is a File of Messages

When the map is run, the first message in the file is found, and converted as specified. Consecutive messages in the file are encountered, and converted. And so on until all of the messages in the file are processed. A burst is identified by the definition of the input type - a message in this case.

If the target is a file, each message produced by a burst is appended to the same file. If you define the target as a message queue, each message produced is placed on the output queue after each burst.

- When the **Source** is a Queue of Messages

Change the source of the input to be a message queue. In this case, fetch the first message on the queue, convert it to the message, and route it to its target - which can be a file or another message queue, for example. Then, fetch the next message on the input queue and process it, and continue to fetch messages one at a time until the input message queue has no more messages.

Retrieval of data prior to the first burst is specific to the adapter.

Batching logical messages

You may process bursts of multiple messages. Under some conditions, this may improve performance. To do this, the logical input (that is, the type of the input card) is a series of messages. Then, adjust the **FetchUnit** to the quantity of messages to process for a given burst.

Automatic Logical Unit Adjustment: If the input type has one component, the range maximum is automatically adjusted for that component to match the **FetchUnit**. For example, the input type is a group with one component (could be a series of messages with a range maximum of 10) and the **FetchUnit** = 20. The series range maximum is automatically adjusted to 20 when the map runs.

If the input type has no components, or has more than one component, the type of that input is not automatically adjusted. In this case, if the amount of data retrieved for a burst does not match the input type definition, the input is either invalid or a burst warning is issued "Input valid, but unknown data found".

Using bursts for large files

To merge three inputs into one, the application takes the first row from the first input, the first row from the second input, and the first row from the third input, and constructs a new row from data of all three input rows. Then, take the second row of the first input, the second row of the second input, and the second row from the third input to construct a new second output row. Then, do the same with each matching input row, constructing a matching output row until all the input is consumed.

To configure the map to run coordinated bursts, set up each input's **CardMode** to **Burst** - and use the same **FetchUnit** for all three inputs.

When more than one input **CardMode** is set to **Burst**, and there is data for one burst and not another, the type definitions for the inputs determine the response. If the card object is required, and that card is set to **Burst**, data must be present if data for any other card set to **Burst** mode is present. In this case, the burst (and subsequently the map) will fail and the error message "One or more inputs is invalid is displayed".

Configuring bursts for error recovery

Errors that occur during burst processing might be recoverable, or they can cause only a burst to fail, or they can be severe enough to cause the entire map to fail. Your card and map settings control many of these failure points.

If the following error occurs...	Card fails	Burst fails	Map fails
Any source connection fails	✓	✓	✓
The FAIL function causes a component rule to fail	✓	✓	✓
The type of an input is invalid	✓	If "stop on first error" is set, input validation fails, causing the burst to fail. If "stop on first error" is not set, input validation continues until all inputs are validated. Input validation as a whole causes a burst failure if any input type is invalid.	If the error limit is reached and a burst fails because one or more inputs are invalid, the map fails.
The FAIL function causes a transformation rule to fail	✓	✓	✓
If an output audit fails		✓	✓
If output audit is used, it is recommended that Scope is set to "burst" or "map".			
If "fail" is set for an input or output Warnings .	✓	✓	✓
Any target connection fails.	✓	✓	✓

If an adapter is referenced as a function argument and no FAIL function is used, all operations rollback when a burst fails. When a burst is successful, any successful operations are committed.

Configuring the RUN function and bursts

When using the RUN function, a source or target used for the map that is run can be configured to be part of the transaction of the burst using the RUN function. To do this, set the **Scope** of the card in the map referenced in the RUN function to **Map**. In this case, if the map that contains the RUN rule fails, any adapter used in the map referenced in the RUN function whose scope is **Map** will exercise the **OnFailure** action.

If the **Scope** of a source or target is set to burst or card, that resource will act based on the appropriate completion status of the card or burst configured in the referenced RUN function.

When using nested RUN functions, if a source or target contained in a nested RUN has a **Scope** set to **Map**, any higher failure will cause that source or target to exercise its **OnFailure** action.

The rollback transaction behavior on maps called with the RUN function is achieved by using an output card, not the PUT function.

Logical or adapter-specified FetchUnits

All adapters have a **FetchUnit**. Some adapters have the capability to aggregate transactional data units. A transactional data unit refers to some unit of data that can be acted upon without affecting the state of another such data unit - for example, removing a message from a queue. Some adapters have the capability to divide one transactional unit into smaller discrete units of data. Some adapters cannot do either of the above. How the **FetchUnit** is interpreted depends on what the adapter's capabilities are.

When the data retrieved from a source adapter represents one transactional unit that cannot be subdivided into smaller discrete units of data without the adapter knowing the content of the data, the **FetchUnit** is interpreted as a logical unit. For example, if an input is a file, it is not a good idea to delete that file until a map completes its work - a stream file is one transactional data unit that cannot be divided into physical units by the adapter. In this case, the **FetchUnit** is interpreted as logical by looking at the type definition of the input data to decide when a burst is complete.

When an adapter can aggregate transactional data units as one source, the **FetchUnit** is interpreted in terms of the transactional data unit. For example, if the data source is MQSeries, multiple messages can be aggregated, and the **FetchUnit** is the maximum number of messages to retrieve for a given burst. In this case, it is up to you to make sure the input type definition matches the amount of data retrieved for a single burst.

For database adapters, an input query represents one transactional unit that can be divided into rows by the adapter. In this case, the **FetchUnit** is the maximum number of rows to retrieve from the database for a given burst.

When the source adapter can only retrieve one transactional unit of data, the source **Scope** is limited to **Map** - this prevents you from trying to delete half a file, or half a buffer, for example. If the source adapter can retrieve multiple transactional units of data, the source **Scope** can be **Map**, **Burst** or **Card**.

Chapter 13. Managing maps

The Map Designer enables work to be reused. Anything that you have done once, you do not have to do again. Input and output cards are managed by copying, deleting, and editing.

Standard windows capabilities

The Map Designer supports standard Windows mouse and keyboard capabilities.

In the following discussion, the word object refers to a graphical object - such as a card in the Navigator - or text - such as text in a map rule.

Moving and copying objects by dragging

Drag procedures allow you to copy and move objects in the Map Designer.

- To *move* an object, drag it to the destination.
- To *copy* an object, press **Ctrl** and drag the object to the destination.

Selecting objects

The Map Designer supports the standard selection capability of Windows.

- To select multiple continuous objects, select the first object, press the **Shift** key, and select the last object.
- To select multiple objects that are not consecutive, press the **Ctrl** key while selecting each object.

In the Navigator, you can select multiple maps, as long as they are in the same source file. You can select multiple cards, as long as they are all input cards or all output cards in the same map.

Deletion

The Map Designer supports Windows' standard deletion capability.

To delete an object, select it and press **Delete**.

Objects that can be deleted with the delete key

The following objects can be deleted by pressing the **Delete** key:

- Maps in the Navigator
- Cards in the Navigator
- Unresolved rules
- Map rules

Cut, copy, and paste

The Map Designer supports standard cut, copy and paste features.

- To cut an object, select it, press the **Ctrl** key and press **X**.

- To copy an object, select it, press the Ctrl key and press C.
- To paste an object, select the destination, press the Ctrl key and press V.

Objects that can be moved by dragging

The following objects can be moved or copied using the drag method:

- Maps in the Navigator - from one map source file to another.
- Cards in the Navigator - from input to output (or output to input) in the same map, or from one map to another.
- Unresolved rules

When you move a card from the output to the input, all map rules on that card are removed.

Objects that can be copied by dragging

The following objects can be copied using the drag method:

- Maps in the Navigator
- Cards in the Navigator
- Unresolved rules

Additional mouse shortcuts

The following list describes other features in the Map Designer:

- To merge a map, select the map in the Navigator, press the Shift key and drag the map name to the destination folder.
- To reorder index occurrences in an output card, press Ctrl and Shift keys while dragging an index occurrence to the desired location.
- To insert a functional map name into a rule, drag the functional map name from the Navigator into the rule bar.
- To insert an item restriction, drag the restriction from the Properties window into the rule bar.
- To copy an existing card within the Navigator, press Ctrl while dragging the card to the desired input card or output card icon within the same map or to another map.

Moving a map

You can move maps from one map source file to another map source file when both maps are open in the Navigator.

To move a map to another source file:

In the Navigator, drag the map icon to the destination map source file.

Copying a map

You can copy maps from one map source file to another map source file.

To copy a map by dragging

In the Navigator, press the Ctrl key while you drag the map icon to the destination map source file.

To copy a map using the copy command

1. Place current focus on the map you want to copy. (The map is active when the name of the map displays in the Map Designer title bar.)
2. From the **Map** menu, choose **Copy**.
The Copy Map dialog box is displayed.
3. In the **Copy to** field, enter a map source file name or browse to display the **Select map file** dialog box in which you can navigate your file system and select the map source file to which you want to copy the map.
4. Click **Open**.
5. Click **OK**.

If you enter a map source file that does not exist, you will be asked if you want the file to be created.

Deleting a map

You can delete maps from map source files, and you can delete map source files.

To delete a map using the Delete key

1. In the Navigator, select the map(s) you want to delete.
2. Press Delete.
A confirmation dialog box is displayed.

To delete a map using the delete command

1. Select the map(s) or map source file(s) to be deleted in the Navigator.
2. From the **Map** menu, choose **Delete**.
A confirmation dialog box is displayed.

Merging a map

You can merge a map from one map source file to another map source file. Merging a map copies the map itself and all of the maps that are related by functional map references to that map.

When you merge a map, if any of the related maps already exist in the target map file, a prompt warning appears, giving you the option to overwrite the existing maps.

To merge a map by dragging

1. In the Navigator, select the map(s) you want to merge.
2. Press Shift, and drag the map name to the folder of the target map source file.

To merge a map using the Merge command

1. Select the map in the Navigator.
2. From the **Map** menu, choose **Merge**.
The Merge Map dialog box is displayed.
3. In the **Map file** field, enter a map source file name.
4. Click **Open**.
5. Click **OK**.

Renaming a map

Maps within a map source file may be renamed. Map source files cannot be renamed.

To rename a map:

1. In the Navigator, select the map to be renamed.
2. From the **Map** menu, choose **Rename**.
3. Enter the new name in the **New Map Name** field.
4. Click **OK**.

Exporting a map source file

Exporting a map source file enables you to store map definitions in XML format in a repository directory. The resulting XML file contains information on map settings, card settings, and map rules.

The **Map** → **Export** command converts a map source file definition from the internal map format to the XML document form.

To export a map source file:

1. Select the map source file you would like to export.
The active map source file name appears in the title bar.
2. From the **Map** menu, select **Export**.
The Export Map(s) to an XML file dialog box is displayed with the active map source filename in the **File name** field.
3. Click **Select**.
The Maps exported successfully confirmation message appears, confirming the selected success export of the map source file.
4. Click **OK**.
The resulting *mapname.xml* file can be opened with a browser or in the Map Designer.

XML format for maps and map rules

Map source files have two definitions:

- XML document form, achieved by exporting a map.
- Internal map source file format, achieved when a map is built and saved in the Map Designer.

The XML open document interface is useful for:

- Storing and retrieving map definitions in a repository
- Printing map definitions
- Storing map definitions in (XML format) text format in version control systems
- Global impact analysis
- Converting business rules from other sources to map rules
- Documentation

The *mapname.xml* file generated with the Export command includes detailed information about the map:

- Map information and map settings
- Input card name and input card settings
- Output card name, output card settings, and map rules

To view the exported XML file in a browser, use the *mapsettings.dtd* and the *mms.dtd* files provided in the *install_dir*\Examples directory. Export maps to the example location or copy the appropriate DTD file to the export map location.

Exported map settings

Exported map settings provide a text version of the map settings compiled in the exported map.

Overrides are used at run time are not included in the exported map settings.

A sample of exported map settings is shown here:

```
<MapAudit Switch="ON">
  <BurstAudit>
    <Data.OnBurstError SizeValidation="WrongSize"/>
    <Execution.Never/>
  </BurstAudit>
  <SummaryAudit execution="OnError"/>
  <SettingsAudit data="OnError" map="OnError"/>
  <AuditLocation.File>
    <Directory.Map/>
    <FileName.Default action="Create"/>
  </AuditLocation.File>
</MapAudit>
```

Exported card settings

Exported card settings provide a text version of the card settings compiled in the exported map. Exported output card settings also contain map rules.

Overrides are used at run time are not included in the exported card settings.

A sample of exported card settings is shown here:

```
<Input>
  <Schema cardnumber="1"
    cardname="ContactFile"
    typetree="Contact.MTT"
    type="ContactFile Data"/>
  <SourceRule>
    <FetchAs>
      <Integral workarea="Create" fetchunit="S"/>
    </FetchAs>
    <GET>
      <FILE_Source>
```

```

        <FilePath>Contact.txt</FilePath>
        <FILE_Source.Transaction OnSuccess="Keep" OnFailure="Rollback" scope="Map"/>
        <Retry Switch="OFF" maxattempts="0" retryinterval="0"/>
    </FILE_Source>
</GET>
<Backup Switch="OFF" When="Always">
    <File_Backup Action="Create">
    </File_Backup>
</Backup>
</SourceRule>
</Input>

```

Exported map rules

Exported map rules provide an XML text version of the map rules compiled in the exported map. Special and reserved characters are shown using XML standards.

The maximum length of a map rule is 32K bytes. The exported size of a map rule may actually be larger if that map rule contains reserved characters. For example, the single quote character (') exports as five characters (").

A sample of exported map rules is shown here:

```

<MapRule rulenum="1">
    <objectset>CityStateZip Field:Label</objectset>
    <objectrule>City Field:Contact:ContactFile + &quot;;, &quot;; +
    State Field:Contact:ContactFile +
    &quot;; &quot;; + ZipCode Field:Contact:ContactFile</objectrule>
</MapRule>
<MapRule rulenum="2">
    <objectset>FullName Field:Label</objectset>
    <objectrule> Last Name Field:Contact:ContactFile + &quot;;, &quot;; + First Name
    Field:Contact:ContactFile</objectrule>
</MapRule>
<MapRule rulenum="3">
    <objectset>Company Field:Label</objectset>
    <objectrule>Company Field:Contact:ContactFile</objectrule>
</MapRule>
<MapRule rulenum="4">
    <objectset>Street Field:Label</objectset>
    <objectrule>Street Field:Contact:ContactFile</objectrule>
</MapRule>

```

Referencing maps

The map reference feature is available for executable maps with only one output card and for functional maps.

Referenced maps must use the same type trees as the referencing map. To reference a map, you simply create a reference from one map source file to a map in another map source file. The referenced map source file remains in its original location.

You cannot alter a referenced map. All card settings, map settings, rules, remarks, and so forth are read-only. If you must change a referenced map, you must either change it from its source location or convert it to a local map.

Version control

For version control purposes, if your map source file (.mms) has references to external functional or executable maps, these maps are not included in the check-in/check-out operation because they are considered read-only, independent

files. To obtain these files, you must retrieve them separately. For example, if you use PVCS Version Manager™, retrieve the files using a "Get" operation.

How to reference a map

Use the following procedure to reference a functional map. To reference a map in this case means to "single-source" - or use a map in an external map source file by creating a reference to it.

For maintenance reasons, use this method as an alternative to copying maps. By referencing a map, maintenance can be performed in one location instead of the several locations that you might have copied the map to.

To reference a map

1. Select the map source file that you want to add a map to.
2. Select **Map** → **Reference**.
The Map Reference Selection dialog is displayed.
3. In the **Referenced Map Source File** field, enter filename and path of the parent map source, or browse to it.
The contents of the selected file are displayed under **Contents**.
4. To view card settings, right-click on a card and select **View** from the context menu.
5. Select one or more maps under **Contents**.
6. To include map dependencies (such as the maps that a functional map references) for the selected maps, enable the **Include map dependencies also** check box.
7. Click **OK** to create the map references.

An alternative way to create a reference to a map is to use a drag-and-drop operation. Use the Ctrl + Alt + Shift key combination and drag the functional or executable map to another map source node. Upon your confirmation, the map converts to a referenced map.

When you view the map settings for a referenced map, you will see the location of the external source file.

To convert a referenced map to a local map

From the Navigator, right-click on the referenced map and choose **Convert to Local Map**.

The map is converted to a local map and is no longer associated with the original source.

If you are using an executable map with only one output card as a functional map, ensure that the inputs and outputs are accessible to it.

Using a resource alias with referenced maps

Using a resource alias for referenced map source files is a three-part process:

- The resource configuration file (.mrc) must be created and defined.
- The Map Designer is configured to locate the .mrc file.
- The resource name-enclosed in percent signs-is used instead of the file path when choosing a map to reference.

For information about creating and defining a resource configuration file (.mrc), see the Resource Registry documentation.

Use the following procedure for using resource alias files with referenced maps in the Map Designer.

To use a resource alias with a referenced map:

1. From the Map Designer menu, select **Tools** → **Options** → **Referenced Maps**.
2. In the **Resource Alias File** field, click browse and locate the resource configuration file (.mrc) to use for referenced maps.
3. Click **OK** to save changes and close the dialog.
4. In the Navigator, select the appropriate map source file.
5. From the Map Designer menu, select **Map** → **Reference**.
6. The Map Reference Selection dialog is displayed.
7. In the **Referenced Map Source File** field, enter the resource name enclosed in percent (%) symbols followed by the name of the map source file you are creating a reference to.

Example: `%resource_name%map_name.mms`

The contents of the map source file appear in the **Contents** pane.

8. Make your map reference selections and click **OK**.
The referenced maps are displayed in the Navigator.

The map settings for the referenced map also indicate that this map is a referenced map using a resource alias.

Chapter 14. Managing cards

You can move, convert, copy, delete, and reorder cards. When using the Navigator to manage cards, use the List view.

Moving a card

You can move a card from the input to the output (or from the output to the input) of the same map.

To move a card by dragging:

In the Navigator, drag the card name(s) to the input card or output card icon in the target map.

When you move a card from the output to the input, all map rules on that card are removed.

Converting a card

Input cards can be converted to output cards and output cards can be converted to input cards.

To convert a card using the Convert command

1. In the Navigator, select the card(s) to be converted.
2. From the **Card** menu, choose **Convert to Input** or **Convert to Output**.

Copying a card

Cards can be copied to a different map source file or to the same map source file. You can copy a card within the same map by adjusting the new map name in the Copy Card dialog box.

To copy a card by dragging

In the Navigator, press the Ctrl key while you drag the card name(s) to the input card or output card icon in the target map.

To copy a card using the copy command

1. In the Navigator, click the **List** tab.
2. Select the card to be copied.
3. From the **Card** menu, choose **Copy**.
The Copy Card dialog box is displayed.
4. Enter the target map source file name in the **File** field or browse your file system and select the source file to which you want to copy the map.
5. Click **Open**.
6. In the **Map** field, enter the name of the map to which you want to copy the selected card or select it from the list.

7. For the **Copy Card As** option, enable either the **Input** or an **Output** option button.
8. If desired, enter the name of the new card in the **Name** field.
9. Click **OK**.

Deleting a card

Input and output cards can be deleted from maps.

To delete a card using the Delete key

1. In the Navigator, click the **List** tab.
2. Select the card(s) you want to delete.
3. Press **Delete**.

To delete a card using the delete command

1. In the Navigator, click the **List** tab.
2. Select the card(s) to be deleted.
3. From the **Card** menu, choose **Delete**.

Editing a card

You can change any of the settings of a card, its name, its type, and its source or target information.

To edit a card:

1. In the Navigator, click the **List** tab.
2. Select the card(s) to be edited.
3. From the **Card** menu, choose **Edit**.
4. Make necessary changes.
5. Click **OK**.

Editing a card

When you edit input and output cards, the map rules that were not affected by the change are retained. You can change anything specified in a map card: the card name, the type tree, the type of the card, and the data source or target.

Changing a card name

When a card name is changed, any reference to that card in any map rule is automatically changed to the new name.

In the following example, the name of the input card is changed from **Input** to **GrowthReport**. After the card name is changed, the new card name, which is referenced in two places in the map rule, is automatically changed.

The input card is edited and the name of the card is changed from **Input** to **GrowthReport**.

The instances of **Input** in the map rule are automatically changed to **GrowthReport**. All instances of the card name are changed in all map rules where it appears.

Changing the type of an input card

You can change the type (**TypeName**) of an input card. Note that when you change the type of an input card, if an object within that card is referenced in a map rule, the object name may now be invalid.

If a map rule contains the name of an object on an input card:

```
Flower:ProgressReading:Input
```

The type of the input card is **GrowthReport** and it is changed to **ProgressReading**, which is a component of **GrowthReport**. The object names in the map rule are now incorrect. **Flower** is a component of **ProgressReading**, which is now the type of the card. After the type of the card has changed, the correct name for **Flower** is **Flower:Input**.

To correct the object name in the map rule, drag the object name to the map rule again. If you change the type of a card and you have many map rules in that card, you may want to use the Replace command in the **Edit** menu instead.

Not all modifications to an input type will affect map rules. For example, if you add a component to an input type, map rules are not affected. Also, if you delete an input component that is not referenced in a map rule, the map rules are not affected.

Changing the type of an output card

You can change the type (**TypeName**) of an output card. Changing the type of an output card retains the map rules for the objects that still exist within the card object, even if they are in a different order.

- If there was a map rule on an object that no longer exists in the card object, that map rule becomes unresolved.
- If there is a map rule on a data object that still exists in the card object, the map rules are retained .

Even though the order of the output components changed, the map rules stay intact.

Changing the type tree of a card

The type tree of a card can be changed. Optionally, you can change the type tree of all the cards in that map and in referenced maps that use the previous tree.

When the **TypeTree** for a card is changed, a confirmation message appears to ask if you want to replace all references to that tree with the new tree.

There are some situations in which you would not want to replace all references to the type tree. For example, if only the input had changed to the definitions in the selected type tree, you would not want the output to change.

Effects of modifying a type tree

When you modify a type tree or any type definitions in that type tree, the next time you open a map that uses that type tree, the map is updated. If the map is currently open when you make changes to the type tree, a prompt appears asking if you want to update the map.

Even if you saved the type tree, but did not make changes to the tree, you are still prompted to update the map.

Changing a type that is used for a card

If you delete, move, or rename a type, and that type is used as the type of a card, the type will appear as having no components. Viewing the **Properties** of the type shows that the type is an **Invalid type**.

Rebuilding a map that has changed

After you have changed a map or updated it because you changed a type tree used in it, build and save the map before running it again.

Chapter 15. Building maps

After you have entered map rules in an executable map, created necessary functional maps, and entered map rules for the desired output data objects, the next step is to build the executable map. A compiled map file is created when no build errors occur when the executable map, and all referenced maps, are analyzed.

An executable map has a data source or target specified for each of its cards. Executable maps are built. Functional maps are not built.

Data sources and targets specified for an executable map are defaults built into the compiled map file. Data sources and targets, and other map settings, can be overridden from the Integration Flow Designer or when a map is run using the Command Server.

Referenced maps

When a map is built, the map and all of the functional maps that are referenced within that map are analyzed. For example, when the map called **MyMap**, which references **MakeRecord**, which references **ItemMap**, is built, all three maps are analyzed. If there are no build errors, a compiled map file is created. The referenced maps are compiled within this file. Referenced maps are not created as separate compiled maps.

Building a map

The Build command on the **Map** menu is used to build the active executable map. The Build command analyzes the logical interfaces within map rules, and if the analysis is successful, compiles the map. Analysis includes checking for missing rules, invalid rules, invalid card definitions, verifying map references, and verifying the arguments of functions. Analysis also looks for circular map references - maps that reference one another. The results appear on the **Build Results** tab in the organizer window.

To build a map

1. In the Navigator, select the map to be built.
2. Select **Map** → **Build**.

The Building Map dialog box is displayed briefly and closes automatically.

The results of the build display on the **Build Results** tab in the organizer window. An empty Build Results page indicates that there were no errors.

To automatically open the organizer window when build errors occur, set that option. From the **Tools** menu, choose **Options**, select **Organizer Window**, and enable the **Show only on errors** check box.

If you build a map and get an error, the error appears on the **Build Results** tab in the organizer window.

The error message shown above refers to an empty rule cell. A map rule is required for each rule cell on an output card.

Edit the output card and enter a map rule. If a map rule is not required, no data transformation must occur for that output object. Use the = NONE map rule.

Using the Organizer window

The Organizer window contains information about a selected map. The information includes unresolved rules, remarks, audit settings, the trace file, the audit log, and build errors. Each tab on the Organizer window can be floating in the main window. This enables you to view the trace file and audit log at the same time.

Viewing and using unresolved map rules

Unresolved map rules appear on the **Unresolved Rules** tab on the Organizer window. An unresolved map rule happens when the type of an output card is changed and there was a map rule on an object that no longer exists in the card object. As a result, that map rule becomes unresolved.

Unresolved map rules are retained for reuse or modification. Unresolved map rules can be dragged to a new output or deleted. There may be times when you have entered a long or complicated map rule that becomes unresolved. Use the drag capability to save time and reduce the risk of errors.

Modifying unresolved rules

Unresolved rules may result from modifying the output type tree such that some output names have changed. You can modify the output names in the organizer window to match the new output names, and use the unresolved rules again. This is useful if there are many unresolved rules on output names that have changed in a similar way. After you modify the output names, the map rules are no longer unresolved.

To modify unresolved rules:

1. Right-click the **Unresolved Rules** tab in the organizer window, and choose **Replace** from the context menu.
2. In the **Find what** field, enter the text to find in the type names.
3. In the **Replace with** field, enter the text to replace it with.
4. Optionally, choose **Match whole word only** and/or **Match case**.
5. Click **Find Next** to find the next occurrence; click **Replace** if you want to replace it, or click **Find Next** again.
6. Or, click **Replace All** to replace all occurrences.

Unresolved rules return to the map rule

If a rule is no longer unresolved, the rule displays in the map rule cell in which it was entered. After you replace all occurrences of **Header** with **Header Record**, the unresolved rules for **Header Record** move back into the appropriate rule cells.

Deleting unresolved rules

Unresolved rules must be deleted before you build a map.

To delete unresolved rules:

1. Select the unresolved rules you want to delete.
2. Press Delete.

Remarks

General comments or comments for particular data objects in a map can be entered on the **Remarks** tab in the organizer window. For example, to make your map user-friendly for another developer enter a comment about a particular map rule. Enter comments about a particular data object by dragging that object into the **Topic** field on the **Remarks** tab.

Entering a remark for an object

To enter a remark for an input or output data object use the object from the card by dragging it to the **Topic** cell on the **Remarks** tab of the organizer window.

To enter a remark for a data object:

1. Drag the data object from the card into a **Topic** cell on the **Remarks** tab.
The data object name appears in the **Topic** cell and the cursor appears in the **Remark** field, ready for you to enter a remark.
2. Type the remark in the **Remark** field.
3. Press Enter.
The object name appears as it does when you drag an object into a rule if the **Use Ellipses** setting is on, the object name appears with ellipses.

Entering a remark for a general topic

You can enter a remark for a topic not associated with a particular object. For example, suppose you want to enter a general remark about when the map should be run. You can enter **When to Run** as the **Topic**, by simply typing it in.

To enter a new remark, click on the last remark and press Enter.

To enter a general remark:

1. Select an empty **Topic** cell.
2. Enter the topic.
3. Enter the remark in the **Remark** field.
4. Press Enter.
There can be only one remark per topic.

Data audit settings

The **Data Audit Settings** tab on the organizer window specifies data objects to be included in the audit log.

To specify that you want to audit a particular object, drag that object from a card into the **Data Audit Settings** tab. Data will only be audited if the **MapSettings** have the **Audit Switch = On** and the **Log > Data = On**.

The name of an object in the **Data Audit Settings** tab is the complete object name. Names are not entered with ellipses, even if the **Use Ellipses** option is on.

Trace tab

To view the trace file for the selected map, click the **Trace** tab on the organizer window or use any text editor to view the *mapname.mtr* file in the specified directory.

The trace file is a text file that records map execution progress. Input data, output data, or both input and output data may be included in a trace file. The trace file should only be used for debugging purposes because performance is impacted.

To create a trace file:

1. Select the map.
2. From the **Map** menu, choose **Settings**.
3. Enable tracing by setting the **MapTrace Switch** to **On**.
4. Specify the desired **MapTrace** settings and click **OK**.

The trace file is created after the map is compiled and executed. After you run a map, the trace file can be viewed on the **Trace** tab in the organizer window.

A trace file of input messages displays a line for each data object, showing:

- The level of the object in the card.
- The offset of the data object in the data stream.
- The length of the data object.
- The component number of the data object.
- The index of the component.
- A portion of the actual data.
- The name of the type to which it belongs.

See "Troubleshooting" for information about using the trace file.

Type tree errors

Each card in the map specifies a type and a type tree. Type trees referenced by the map must be analyzed and saved. If you build a map, and a type tree referenced by that map has not been analyzed, or it has been analyzed but produced at least one error, an error message to that effect appears. For example, if the type tree address.mtt was not successfully analyzed and it is used in a map, building that map causes a warning message to be displayed.

You are not prevented from building the map. However, you may get unpredictable results when running the map. For example, if a group's components are not distinguishable, it is not clearly defined where one component ends and the next one begins. The data may not validate correctly.

A type tree that has not been analyzed as error-free includes the following:

- A type tree that was analyzed and produced at least one error.
- A type tree that has been analyzed, but not saved.

Map build analysis errors

If a build error occurs, a map cannot be compiled until the problem is corrected. Build errors are written to the **Build Errors** tab on the Organizer window and to a map build analysis results file with the .mme file name extension. This .mme file is written to the folder where the map source file is located.

If build errors occur, view the errors, correct the problem(s), and build the map again. A map cannot be run if it has not been built.

See "Error and Warning Messages" for a complete list of map build analysis messages.

Build results of a map provide information on errors and warnings that occur during the map build process. Errors and warnings are reported on the Build Results tab of the Organizer window.

During the map build process, the map and all of the functional maps that are referenced within that map are analyzed.

- Build errors prevent a map from being compiled
- Build warnings alert you of a condition that might be problematic.

You can ignore warnings to allow the map to be built even if there are warnings.

If you build a map and there are warnings, but no errors, the map is compiled.

Viewing build results

Build results are written to a map build analysis results file with the .mme file name extension. This .mme file is written to the folder where the map source file is located. The .mme file is a text file and may be viewed with a text editor or from within the Map Designer.

To view build results:

1. From the **Map** menu, choose **Organizer**.

The Organizer is displayed.

2. Click the **Build Results** tab.

The organizer window can be configured to automatically open if there were build errors. Enable the **Tools Options** → **Organizer** option for **Show only on errors**.

An empty **Build Results** tab indicates that there were no map build errors.

Searching in the Organizer

To search for text on the **Trace**, **Audit Log**, or **Build Results** tab, right-click on the desired tab and choose **Find** from the context menu.

Map build warning messages

A build warning alerts you about a potentially problematic condition. If you build a map with warnings, but no errors, the map is successfully compiled. You can ignore warnings.

To ignore warnings:

1. From the **Tools** menu, choose **Options**.
2. Select **General**.
3. Enable the **Suppress build warnings** check box.

Refer to "Map build warning messages" for a complete list of map build warning messages.

Map build compile errors

Maps with compile errors may not be built. Maps that are not compiled may not be run.

For example, a compile error results when a map is built and the input contains a delimited group whose delimiter has not been specified. To correct this compile error, specify the delimiter in the type tree, analyze it, and save the type tree. After saving the type tree, build the map again.

See “Map build compile errors” for a complete list of map build compile error messages.

Viewing build results

Errors or warnings that occur during the map build process appear on the **Build Results** tab in the organizer window.

To view the build results:

1. Select the map whose build results you want to view.
2. From the **Map** menu, choose **Organizer**.
3. Click the **Build Results** tab.

Compiled map files

If no errors occur, invoking the Build command creates a compiled map file. The compiled map file is written to the location of the map source file (.mms). The name of the compiled map file is the name of the map that was built, with the .mmc filename extension. For example, the **MyMap** map creates MyMap.mmc.

Note: If you have more than one executable map with the same name, make sure that you save the different map source files in different folders.

A compiled map file is an input for the Command Server or Launcher. Running a map from a command line uses the compiled map file.

Building a map for non-windows platforms

To run a map on a non-Windows platform, you must build the map for that platform using the Build for Specific Platform command. To run a map on a non-Windows platform after it is built, that platform must have the Command Server installed.

Maps compiled for specific platforms are compiled with platform-specific file name extensions. The name of the platform-specific compiled map file is the executable map name with the platform-specific extension, such as .aix. For example, building the map “MyMap” for the Linux platform creates the file MyMap.lnx in the map directory.

These default file name extensions prevent you from inadvertently overwriting your original compiled map and help identify the compiled map file to transfer to the specific platform environment.

When deploying systems that contain maps with XML types to non-windows platforms, the XML Schema and XML input files must be transferred in binary format.

Platform-specific file name extensions

Maps built for execution on Windows platforms use the default .mmc compiled map filename extension.

The Build for Specific Platform command creates a compiled map file in the format required for a given platform, which accounts for byte-order and character set differences on that platform.

By default, the Map Designer assigns a platform-specific filename extension to maps built for specific platforms. However, you can change the compiled map filename and extension in the **File Name** field of the Select Platform dialog box as shown:

The list of default filename extensions for compiled maps are listed in the following table.

Platform	Extension
HP NonStop	.nsk
HP-UX™ (Itanium processor)	.hpi
HP-UX™ (PA-RISC processor)	.hp
IBM AIX™ (RS/6000 processor)	.aix
IBM™ z/OS	.mvs
Linux (Intel processor)	.lnx
Microsoft Windows	.mmc
Sun Solaris™ (SPARC processor)	.sun

Create a folder exclusively for platform-specific maps.

To build a map for a specific platform:

1. Select the map you want to build.
2. From the **Map** menu, choose **Build for Specific Platform**.
The Select Platform dialog box is displayed.
3. For the **Platform**, select the platform on which you plan to run the map.
The **File Name** field on the Select Platform dialog box updates with the name of the compiled map that will be ported to that platform.
4. Click **OK**.

After the map is built for the specific platform, perform a binary file transfer of compiled map file to the Command Server environment. For example, in a UNIX environment, you might use FTP to transfer the ported map to your UNIX server.

- A platform-specific build is not required if all of the items referenced in map rules have specific character sets and a specific byte order. (No items have a native definition.)
- For example, you may have a map whose only purpose is to convert from ASCII to EBCDIC. However, to avoid mistakes, always use the Build for Specific Platform command when your target platform has a different byte order or character set than the one in your map development environment.
- A platform-specific build is not required if you select a platform whose byte order and character set are the same as those on your development machine. If you do select one of these platforms, a message indicates that this platform does not require a platform-specific build. In this case, the compiled map file will have the extension .mmc.

Chapter 16. Running a map

After a map has been compiled, it can be run. When you run a map, the information in the compiled map file is used to create the output data.

To run a map, use the Map Designer, Launcher, a transformation server, or an API. The Design Studio includes a Windows Command Server. You might also have a Command Server for another platform, such as AIX, Sun Solaris, HP-UX, HP NonStop, z/OS, or Linux. Some Command Servers have different command options. For information on command options for a Command Server in a particular environment, see the Command Server documentation.

If you have an Launcher, you use the Integration Flow Designer to configure maps to run. You can run an WebSphere Transformation Extender map from within an application using an WebSphere Transformation Extender programming interface (TXPI). And, you can use the RUN function to run a map within another map.

Running a map

You can run a map from within the Map Designer.

To run a map:

Select the map you want to run and do one of the following:

- From the **Map** menu, choose **Run**.

The Command Server window is displayed.

The Command Server window displays the progress of the map execution, the total number of input objects found, and the total number of output objects built. These statistics are shown to indicate progress the data is mapped.

Default map settings are set in the Run Options dialog box. After a map is created, select **Settings** from the **Map** menu to change the settings for the selected map.

Map error and warning messages

Map Designer error and warning messages provide information about map execution return codes, map build errors, map compile errors, and warning messages, and run time map return codes.

Run results

Run results are the input data used in the map and the output data generated by the map. The run results are the actual data as described by the data objects defined in the input and output cards.

Do not confuse run results with build results, which provide information about errors and warnings that occur during the map build process.

Viewing the map run results

After you run the map, you can view the run results. The state of the run results are remembered for each map, and run results may be viewed with shortcut access methods.

To view run results:

1. Select the map whose run results you want to view.
2. From the **Map** menu, choose **Run Results**.
The Run Results dialog box is displayed.
3. Disable the check box next to the inputs and outputs you do not want to view.
4. Click **OK**.

Each data source and target appears in its own window.

Run results display options

When displaying run results for input and output data, you can view the results as XML or hexadecimal (hex) in addition to the regular text format.

Run results for data files with the following filename extensions display in XML mode by default: .xml, .xsd, .xsx, .wsdl, and .xsl. If the data files do not have the mentioned filename extensions, but an XML prolog (<?xml) is present at beginning of the data, the run results are also displayed in XML mode. Otherwise, the results display in a general text format or in hex mode if selected.

If XML validation errors occur, they are displayed as text.

To view run results in hex mode:

In the Run Results dialog box, enable the **Display in Hex Mode** check box.

If you want the offset to be displayed by the decimal value, from the **Tools** menu, choose **Options**. For the **Run Results**, enable the **Show offset in decimal values** check box.

To view run results data as XML:

1. When viewing the data source and target run results, right-click on the display window to view a context menu.
2. Choose **XML Display**.

If you select **XML Display** and the data file does not meet the specified criteria (listed under "Run results display options") an error message is displayed.

Closing Run Results windows

You can close multiple Run Results windows at once.

To close all run results windows for a selected map

From the **Map** menu, select **Close all run results**.

To close all data windows

From the **Window** menu, select **Close All**.

Viewing run results with shortcut access methods

You can view run results for the selected map.

To view run results for the selected map:

1. Select the map in the Navigator.
2. Press the **Shift** key and select **Map** → **Run Results**

The run results are shown without displaying the **Run Results** dialog box.

The state of the run results that are shown in a are saved (in a *mapname.mopt* file in the map directory) to enable viewing at a later time.

Chapter 17. Map performance

You can use the Map Profiler to analyze map execution behavior.

Map Profiler overview

The Map Profiler is a user-configurable utility that captures and reports map execution statistics. The profiler focuses on component and mapping rules and the functions and types within those rules. The resulting information enables you to see where performance is lagging and in turn make improvements in your maps.

For example, in the profile report you see that the processing time for a LOOKUP function is significantly more than other functions. You can modify the rule to use a better choice, such as the SEARCHUP function (because in this specific case the data in the LOOKUP is ascending), and as a result the processing time is greatly reduced.

Map Profiler output

The profiler can capture the following statistics:

- Amount of time spent processing component and mapping rules and subordinate functions.
- Number of times rules and functions are executed (from validation through output).
- Number of times type objects are accessed.
- Depth (nesting level) of the object being profiled.

Type names

Similar to the map trace file, the type names referred to in the profile output represent the full type name in the original type tree, not that of the type object that is visible in the Map Designer.

Function times

The processing time of any subfunctions rolls up into the processing time of the top level function. For example, if an EITHER function contains an IF function, the recorded time for the EITHER function would include the processing time of the IF function.

RUN maps

When a parent map contains a RUN map, the RUN map is not profiled. To profile a RUN map, it must be done outside of the RUN rule. (You can do this by running it as a top level executable map.)

Configuring the Map Profiler

The profiler can be configured and invoked from the Map Designer interface or from the command line.

From the Map Designer interface, you can enable or disable the profiler from the Map menu or the toolbar. Before running the map, you must open the Map

Profiler options and enable either the **Function Times** option or **Type Times** option to see statistics in the output (otherwise the report is empty).

Command line

The dstxprof console application enables you to profile maps from the command line.

See the Utility Commands documentation for information about using the dstxprof utility.

Using the Map Profiler

To profile a map:

1. Choose a map and select **Map** → **Enable Profiler**.
2. Build and run the map.
3. Open the Organizer and select the **Profile** tab to view results.

You can also open Map Profiler results in a text editor from the map directory. The filename is a combination of the map name and a "PROF" extension. For example, *map_name.MPROF*.

Profiler output example

A full report containing both function and type information looks similar to the following example:

```

Individual Function Breakdown
function          depth iterations    time    map          type
-----          -
validation
VALIDATE_CARD      0         1       170  Executable
VALIDATE_CARD      0         1       189  Executable
TYPE_NAME          1        16         4  Executable

Detail Record Input Data
mapping
BUILD_CARD         0         1       197  Executable
MAP_NAME           1         1         64  Executable      PO Data
TYPE_NAME          2         1          0  Executable      PO Data
CHOOSE             2         1         11  Executable      PO Data
TYPE_NAME          3         1          0  Executable      PO Data
INDEX              3         1          7  Executable      PO Data
TYPE_NAME          2         1         13  MakePO          Company Field Data
ADD_OP             2         1          1  MakePO          PO# Field Data

Type per Rule Breakdown
time    map          type
----    ---          ----
validation
  359    Executable

mapping
   4    Executable      Detail Record Input Data

mapping
  197    Executable
   82    Executable      PO Data
   13    MakePO          Company Field Data
    1    MakePO          PO# Field Data

```

0	MakePO	PODate Field Data
19	MakePO	Detail Record Data
14	MakeDetail	Description Field Data

Profiler summary report example

A summary report containing both function and type information looks similar to the following example:

```

Individual Function Breakdown
function          depth iterations  time    map      type
-----
validation
VALIDATE_CARD    0         1       208    Executable
VALIDATE_CARD    0         1       192    Executable
TYPE_NAME        1         16        4    Executable  Detail Record Input Data

mapping
BUILD_CARD       0         1       193    Executable
MAP_NAME         1         1        59    Executable      PO Data
TYPE_NAME        2         1         1    Executable      PO Data
CHOOSE           2         1         4    Executable      PO Data

Type per Rule Breakdown
time    map      type
----    ---    ----
validation
400      Executable

mapping
4        Executable      Detail Record Input Data

mapping
193     Executable
65      Executable      PO Data
16      MakePO          Company Field Data
1       MakePO          PO# Field Data
0       MakePO          PODate Field Data
18      MakePO          Detail Record Data

Type per Rule Summary
iterations  time  map      type
-----
validation
1          4    Executable      Detail Record Input Data

mapping
3          140 Executable      PO Data
3          16  MakePO          Company Field Data
3          3   MakePO          PO# Field Data
3          0   MakePO          PODate Field Data

Function Summary
function          iterations  time
-----
VALIDATE_CARD    2          400
BUILD_CARD       1          193
INDEX            3           1
TYPE_NAME        73         49
STRING_LITERAL   15         3

```

For best results...

- If you have a large amount of data, the profiler output could be very large or unmanageable. For this reason, it is best to profile only a subset of the original data.
- You can reduce profiler output by specifying a time limit so that nothing is reported until it exceeds the time limit that you specify. See section "Configuring the Map Profiler" for details.
- When running the profiler, always limit or terminate other processes running on your machine to avoid adverse results.

Chapter 18. Managing invalid data

You can map invalid data to an output, just like any other data. For example, you can place all invalid data in a separate output for editing or debugging purposes. You can map invalid data to an error report.

To map invalid data, it must be contained in a valid object. A map does not run unless all input card objects are valid. For example, if you want to map an invalid purchase order in a file of many purchase orders, the file itself must be valid.

Using the Restart Attribute

To map invalid data of a particular object, assign the Restart attribute. Errors must be ignored for invalid data object during validation. You can then map the invalid data using any or all of the error functions REJECT, ISERROR, and CONTAINSERRORS. For a detailed discussion of the Restart attribute, see the Type Designer.

Using the REJECT function

To map invalid data, use the REJECT function. The REJECT function is used in conjunction with the restart attribute. The REJECT function operates on an invalid data object contained within a valid object. The output of the REJECT function is a series of text items.

The syntax of the REJECT function is the following:

```
REJECT (component_with_restart_attribute)
```

Use the REJECT function with expressions that map valid data. For example, you can generate an error report that contains the invalid input data, in addition to the valid company name and date from the input.

Using the ISERROR function

You may conditionally map data based on whether or not a certain object is invalid using the ISERROR function. The ISERROR function has one input argument and returns a boolean value. It returns "true" if the argument is invalid or returns "false" if the argument is valid.

The ISERROR function is used with the Restart attribute. The ISERROR function operates on an invalid data object contained within a valid object.

In this input file of customers, the third customer record is invalid.

```
Gisela,1252 S. Broward/Ft. Lauderdale,523-2622,22
```

```
Dede,1513 Palatine Rd./Boca Raton,252-6560,86
```

```
Lewis,74099 S. 67th Ave/Ft. Lauderdale,332-8665,3.3
```

```
Eric,6933 Main St./South Miami Beach,291-7281,56
```

To generate a report that indicates whether each customer object is valid, assign a Restart attribute to **Customer(s)** in the input.

The output report is composed of **Line(s)**, where **Line** is defined as a text item. The map rule on **Line** uses the ISERROR function to check whether **Customer** is invalid. If **Customer** is invalid, the result is "INVALID Customer Data" and the REJECT of **Customer**. If **Customer** is valid, the result is "GOOD Customer Data" and the TEXTconversion of the **Customer** object itself.

Using the CONTAINSERRORS function

You may not want to map the input data if it contains any invalid data. In this example or situation, use the CONTAINSERRORS function. The CONTAINSERRORS function has one input argument that is the name of the object you want to test. The CONTAINSERRORS function returns "true" if the argument contains any invalid object or returns "false" if the argument does not contain any invalid object.

The CONTAINSERRORS function only operates on a valid object.

For example, if you have the same input file as in the example above, the third **Customer** record is invalid. You are mapping the input to the output file of **CustomerEntry(s)**. If the entire input contains any invalid object, do not produce any output. If the entire input is valid, map the **Customer(s)** to **CustomerEntry(s)**, using the functional map **MakeEntry**.

The map rule on **CustomerEntry(s)** uses the CONTAINSERRORS function to check if the entire **List** contains any errors. If **List** contains an error, no output is produced. If **List** does not contain errors, the **Customer(s)** will be mapped to **CustomerEntry(s)**.

The result is no output, because **List** does contain an error - the third **Customer** record is invalid.

Chapter 19. Map audit overview

A map audit log is created when a map is executed. The log information is based on a particular map and provides information about map execution, map settings, card settings, and specific data objects. Map audit logs are in XML format and the contents can be mapped in such a way as to provide performance statistics and information about the content of your data.

Audit log contents

A map can be configured to produce a map audit log consisting of the following:

- Execution statistics in summary or burst mode
- Burst information
- Map settings
- Data settings
- Data content information

Settings that control audit log content

- The map settings that control the map audit are defined by accessing the map settings.
- The **Data Audit Settings** tab on the Organizer window defines specific data objects to be included in the audit log.

You can also specify audit options from the command line. (Refer to Execution Commands documentation for command options.) The audit log can be stored in a file or returned in memory if you use the RUN function or an API.

Viewing the audit log

If you have configured your map settings to include an audit log, you can view the audit log results from the Organizer window.

To view the audit log:

1. After the audit is configured in the map settings and the map is executed, select the map in the Navigator.
2. From the **Map** menu, choose **Organizer**.
3. Click the **Audit Log** tab.

If the value of the **AuditLocation** map setting is **File**, the audit log file can also be viewed with a text editor. The file, *mapname.log*, is output to the map directory.

If the value of the **AuditLocation Filename** is **Unique**, the audit log cannot be viewed from the Organizer window.

Execution summary in the audit log

The **MapAudit Switch = On** and the **SummaryAudit Execution = On** map settings produce an audit log with execution summary information. The **ExecutionSummary** element of the audit log provides performance data for the map as a whole.

Execution summary information includes:

- Map status information
- Compiled map name
- Map return code
- Number of input objects found
- Elapsed execution time
- Number of output objects built
- Burst restart count
- Source data report by card
- Map return code message
- Target data report by card
- Work area information by card

The format of the elapsed time is presented differently on Windows platforms than on non-Windows platforms.

Some information is optional, and does not always appear in the **ExecutionSummary** element of the audit log. For example, information about objects found and built is provided only when using a command engine. **WorkArea** data is provided only when a work area is used during map processing.

Elapsed execution time in the audit log

The execution audit log records the elapsed time. The elapsed time format is presented differently on Windows platforms than on non-Windows platforms.

- On Windows platforms, the elapsed time of map execution is described to four decimal places.
- On non-Windows platforms, the elapsed time of map execution is presented in whole seconds only (rounded).

Note: When using the `tryaudit.mtt` type tree from the examples (`install_dir/examples`) to read map audit logs, modify the type tree appropriately for the execution platform.

Source data report by card

In the audit log, the source data report by card provides information on:

- Value of the adapter **Source** card setting
- Source data byte count
- Adapter return code and message
- Adapter command line or file path
- Data source time stamp

Target data report by card

In the audit log, the target data report by card provides information on:

- Value of the adapter **Target** card setting
- Target data byte count
- Adapter return code and message
- Adapter command line or file path
- Data target time stamp

Work area information by card

In the audit log, the work area information includes:

- Work area type (file or memory)
- Work area file name and time stamp if applicable
- Work area byte count

Execution summary audit log sample

The following is a sample of the execution summary portion of the audit log.

```
<ExecutionSummary MapStatus="Valid" mapreturn="0" ElapsedSec="0.0373"
BurstRestartCount="0">
  <Message>Map completed successfully</Message>
  <CommandLine>'install_dir\Examples\CallsSummary.mmc'</CommandLine>
  <ObjectsFound>18</ObjectsFound>
  <ObjectsBuilt>12</ObjectsBuilt>
  <SourceReport card="1" adapter="File" bytes="52" adapterreturn="0">
    <Message>Data read successfully</Message>
    <Settings>install_dir\Examples\stores.txt</Settings>
    <TimeStamp>18:10:04 December 26, 2000</TimeStamp>
  </SourceReport>
  <SourceReport card="2" adapter="File" bytes="69" adapterreturn="0">
    <Message>Data read successfully</Message>
    <Settings>install_dir\Examples\CALLS.TXT</Settings>
    <TimeStamp>18:10:04 December 26, 2000</TimeStamp>
  </SourceReport>
  <TargetReport card="1" adapter="File" bytes="119" adapterreturn="0">
    <Message>Data written successfully</Message>
    <Settings>install_dir\Examples\summary.txt</Settings>
    <TimeStamp>10:14:34 April 27, 2001</TimeStamp>
  </TargetReport>
  <WorkArea type="File">
    <inputarea card="1" Path="install_dir\Examples\CallsSummary.I01"
      TimeStamp="10:14:34 April 27,
      2001" bytes="65695"/>
    <inputarea card="2" Path="install_dir\Examples\CallsSummary.I02"
      TimeStamp="10:14:34 April 27,
      2001" bytes="65695"/>
    <outputarea card="1" Path="install_dir\Examples\CallsSummary.001"
      TimeStamp="10:14:34 April
      27, 2001" bytes="65695"/>
  </WorkArea>
</ExecutionSummary>
```

BurstAudit execution in the audit log

The **MapAudit Switch = On** and the **BurstAudit Execution = On** map settings produce an audit log with burst execution information. The burst element provides information on the performance of the burst.

- If all input cards have the **FetchAs** mode of **Integral**, there is one burst element in the audit log.
- If any input card has the **FetchAs** mode of **Burst**, there is one element for each burst.

The execution log contains the return code of the burst, the amount of time it took to process the burst (in seconds), and a status of each input and output. The input and output status include the card number, the bytes processed, the adapter return code, and the content return code.

For example, if a burst fails because an adapter could not connect, the audit log defines which adapter failed. If an input is invalid, the content return code specifies which input is invalid. If an output rule fails, the content return code tells you which output failed.

A sample of the burst audit execution information in the audit log is:

```
<Burst count="1">
<ExecutionLog burstreturn="0" ElapsedSec="0.0118">
<inputstatus card="1" bytes="52" adapterreturn="0" contentreturn="0"/>
<inputstatus card="2" bytes="69" adapterreturn="0" contentreturn="0"/>
<outputstatus card="1" bytes="119" adapterreturn="0" contentreturn="0"/>
</ExecutionLog>
</Burst>
```

BurstAudit Data in the audit log

The **MapAudit Switch = On** and the **BurstAudit Data = On** map settings produce an audit log with burst audit data information. The burst audit data information includes a burst count, the card number, the object index, the audit log status code, the object level, the size of the object, and the object name.

A sample of the burst audit data information is:

```
<Burst count="1">
<DataLog>
<output card="1">
<object index="1" level="1" size="24" status="V00">Text1 Field</object>
</output>
</DataLog>
</Burst>
```

Data content information in the audit log

You can audit input and output data. The data audit settings determine what information about the data is written to the audit log.

Auditing data keeps track of specified objects in the input(s) and/or output(s) of a map. The burst audit data information in the audit log can track every occurrence of a record in a file and record the total count of records.

In the resulting audit log, the **<DataLog>** section follows the **</Burst>** tag. Within **<DataLog>**, there is an entry for the first input card. Within the input entry, there is an entry for each occurrence of the object.

If item data is specified in an audit setting, and the item is to be interpreted as binary, the line following the object line has the start tag **<Hex>** and the data is presented as pairs of hex digits for each byte.

In the audit log, the **<DataLog>** section follows the **</Burst>** tag. Within **<DataLog>**, there is an entry for the first input card. Within the input entry, there is an entry for each occurrence of the object.

If item data is specified in an audit setting, and the item is to be interpreted as binary, the line following the object line has the start tag **<Hex>** and the data is presented as pairs of hex digits for each byte.

Auditing data content

You can audit input and output data. The data audit settings determine what information about the data is written to the audit log.

To include data information in the audit log:

1. Select the map for which you want to create an audit log.
2. From the **Map** menu, choose **Settings**.
3. For the **MapAudit** setting, set the **Switch** to **ON**.
4. For the **BurstAudit Data** setting, select **ON**.
5. Click **OK**.

After you run the map, you can view the audit on the **Audit Log** tab in the **Organizer**.

Data settings in the audit log

The **MapAudit Switch = On** and the **SettingsAudit Data = On** map settings produce an audit log with input and output card setting information. Data settings are defined on the input and output cards of a map and include:

- Card number
- Input card mode (as defined with the **FetchAs** card setting)
- **WorkArea** reuse setting
- Backup card settings
- Source and target adapter settings
- Transaction settings
- Resource details
- Retry settings

Data settings in the audit log sample

```
<DataSettings>
<InputData card="1" CardMode="Integral" WorkArea="!Reuse">
  <Backup switch="OFF"></Backup>
  <Source adapter="File" OnSuccess="Keep" OnFailure="Rollback" Scope="Map"
Warnings="Ignore">
    <Resource>install_dir\Examples\stores.txt</Resource>
    <Retry switch="OFF"></Retry>
  </Source>
</InputData>
<InputData card="2" CardMode="Integral" WorkArea="!Reuse">
  <Backup switch="OFF"></Backup>
  <Source adapter="File" OnSuccess="Keep" OnFailure="Rollback" Scope="Map"
Warnings="Ignore">
    <Resource>install_dir\Examples\CALLS.TXT</Resource>
    <Retry switch="OFF"></Retry>
  </Source>
</InputData>
<OutputData card="1">
  <Backup switch="OFF"></Backup>
  <Target adapter="File" OnSuccess="Keep" OnFailure="Rollback" Scope="Map"
Warnings="Ignore">
    <Resource>install_dir\Examples\summary.txt</Resource>
    <Retry switch="OFF"></Retry>
  </Target>
</OutputData>
</DataSettings>
```

Map settings in the audit log

The **MapAudit Switch = On** and the **SettingsAudit Map = On** map settings produce an audit log with map setting information.

Map settings include:

- Map audit settings
- Map trace information
- Paging scheme
- Validation specifications
- Audit log location
- Work space information
- Century information

Map settings are viewable in the audit log:

```
<MapSettings>
<MapAudit switch="ON">
  <Log execution="OFF" data="OFF" mapsettings="ON" datasettings="OFF"/>
  <Location type="File" action="Create">
    <Directory type="Map">install_dir\Examples</Directory>
    <FileName type="Default" prefix="MapName">CallsSummary.log</FileName>
  </Location>
</MapAudit>
<MapTrace switch="OFF"></MapTrace>
<WorkSpace>
  <Location type="Memory">
  </Location>
  <Paging>
    <PageSize>64</PageSize>
    <PageCount>8</PageCount>
  </Paging>
</WorkSpace>
<SlidingCentury switch="OFF"></SlidingCentury>
<CustomValidation switch="OFF"></CustomValidation>
<Retry switch="OFF"></Retry>
<BurstRestart switch="OFF"></BurstRestart>
<Warnings type="Every" action="Warn"></Warnings>
</MapSettings>
```

Auditing a map

A map audit log contains information about a particular map.

To create an audit log:

1. Select the map.
2. From the **Map** menu, choose **Settings**.
3. Specify creation of an audit log by setting **MapAudit Switch = ON**.
4. Specify the desired **BurstAudit**, **SummaryAudit**, **SettingsAudit**, and **AuditLocation** settings.
5. Click **OK**.

The audit log file is created as specified on the next map execution. After you run a map, the audit log file can be viewed on the **Audit Log** tab in the **Organizer** or from a text file (*map_name.log*) in the map directory.

Mapping from the audit log

Mapping information from the audit log can provide useful performance statistics and information about the content of your data. For mapping purposes, the `tryaudit.mtt` type tree provided in the **examples** directory defines the audit log data objects.

The format of the audit log is XML-based to enable use with other applications. The audit log is defined as a series of **MapAudit** elements. Each **MapAudit** element represents the log data for one map instance. You can get audit data for many map instances when the **AuditLocation** map setting for **FileName Action** = **Append**.

The **MapAudit** general element in the input card shown below displays the sequence of the information in the audit log.

ExecutionSummary in the Tryaudit type tree

The audit log is created for a specific map. During map execution, after all bursts complete, the status of the map is summarized in the **ExecutionSummary**. The **ExecutionSummary** element in the audit log provides performance data for the map as a whole. This information appears in the audit log when the **SummaryAudit** → **Execution** map setting = **ON**.

Open `install_dir\examples\general\audit\tryaudit.mtt` to view the structure of **ExecutionSummary**.

The **ExecutionSummary** element contains:

- Map status
- Map return code
- Amount of map processing time
- **BurstRestartCount** indicator of how many times a burst restart occurred
- Return message for the map
- Command line that was used to re-configure any options compiled in the map
- Report for each source and target
- Work-area data

Summary information provides performance data for the map as a whole. For example, summary information might include the fact that all bursts succeeded, but the map failed because **Scope** was set to **Map**, and an adapter failed.

Summary source and target reports the adapter used for each, the total number of bytes processed over all bursts, and the final adapter return code.

Configuring data audit

Use the **Data Audit Settings** tab in the Organizer window to specify what information about the data to include in the audit log. Audit logs are for a single map.

To specify that you want to audit a particular object, drag that object from a card into the **Data Audit Settings** tab.

To audit a particular data object:

1. For the specific map with cards that contain the data object, define the **MapAudit** map settings in the Map Settings dialog box:
 - **MapAudit > Switch = On**
 - **BurstAudit > Data = On**
2. From the **Map** menu, choose **Organizer**.
The **Organizer** window is displayed.
3. Drag the data object from the card into the **Audit** field on the **Data Audit Settings** tab.
4. From the **Track** list, choose what information to track.
5. From the **Detail** list, choose what information to report about the object's nested components.
6. If possible, from the **Item Data** list, specify when to show the item data of that object.

Data audit settings are compiled into the map file. If you change the audit settings in a map, you must build the map again for those settings to take effect.

The name of an object in the **Data Audit Settings** tab is the complete object name. Names are not entered with ellipses, even if the **Use Ellipses** option is enabled.

Detail

In the **Detail** list on the **Data Audit Settings** tab in the Organizer window, specify what information to include about an object's nested components.

Value Description

Occurrence

A line appears in the audit log for each occurrence of that object of each nested component.

Warning

A line appears in the audit log for any nested component that causes a warning.

Error A line appears in the audit log for any nested component that causes an error.

None No information about nested components is included for that particular object.

Item data

In the **Item Data** list on the **Data Audit Settings** tab in the Organizer window, specify what actual *data* to show in the audit log. This applies to all items nested within the tracked object.

Value Description

Occurrence

The data of each nested item within that object appears in the audit log.

Warning

The data of each nested item within that object that causes a warning appears in the audit log.

Error A line appears in the audit log for each nested item within that object that causes an error.

None No information about nested components is included for that particular object.

Status codes in the audit log

The status attribute in the Audit Log is a letter followed by a code number. The letter of the status attribute tells whether the data is valid V, or caused an error E, or caused a warning W.

Elements in the data audit log

The audit log is in XML format. The tryaudit.mtt type tree in the **examples** directory describes the audit log data.

When **BurstAudit Data** map setting is **ON**, and there is audit data, each burst in the audit log has a **<DataLog>** element. General data elements are defined as **ZZ_Other DataLog**.

Chapter 20. Troubleshooting

There are various methods of troubleshooting that can assist you in situations where when running a map, the following types of errors occur regarding input or output data:

- Data is invalid
- Data contains invalid objects
- Unknown data exists

About data validation

When you run a map, the first thing that happens is the input data is validated by comparing the data to its definition in the type tree. After validating data, the output card map rules are evaluated, and the output objects are built.

If you run a map and invalid or unknown input data is encountered, you can analyze what happened during the validation process. In addition, you can analyze the output data.

Overview of debugging aids

If you run a map and get a runtime error, or you do not get the expected output, use any or all of the following debugging tools:

- Command Server window
- Run results
- Audit Log
- Customized validation settings
- Map Debugger
- Trace files

Command Server window

After a map is run, a runtime message appears in the Command Server window. This message provides information about the map.

For example, the message "Map completed successfully" indicates that all inputs are valid. However, you might see the message "One or more inputs was invalid"-indicating that an input card object is invalid.

The message "One or more inputs was invalid" can occur even if you are absolutely sure that the input data is valid. In that case, there may be an error in the data definitions in the Type Designer. For example, if you mistakenly defined an item to be an integer but the data is actually a decimal number, that data object is considered invalid because it does not match its type definition. In general, when data is diagnosed as "invalid", either the data is invalid or the type definition is incorrect.

The Command Server window also displays the progress of the data being mapped. For example, you can see when input objects are found and when output objects are built.

Run results

Run results are the input data used in the map and the output data generated by the map. The run results are the actual data as described by the data objects defined in the input and output cards.

By viewing the results of a map execution, you can gain important information about how the data was mapped. For example, when no output objects are built, your input data might be invalid. Or, if you see only a portion of the output data, your input data might be valid but it could contain unknown data.

For more information about run results, refer to "Run results" on page 1-2.

Audit log

The audit log stores the information about your data that you specify from within the map settings.

For example, you can specify that the audit log only records failure information.

For a details about using the audit log, see "Map Audit".

Customized validation settings

Another way to troubleshoot a map is to customize the validation settings. Validation can be set to ignore certain aspects of the data during validation by setting custom options in the **Validation** map settings.

For example, you can choose that the validation process ignores item restrictions. This means that even if an item has restrictions defined, the restrictions are not considered when that item is validated.

Map Debugger

The debug functionality in the Map Designer can assist you in creating and troubleshooting maps. Use the Map Debugger when you have narrowed a problem down to a certain object or function call. For example, when you know that an object is not producing the correct output.

The map debug functionality allows you to break the execution of a map and inspect the current object values (both input and output objects) used within the currently executing output rule.

You cannot use the Map Debugger when the Map Profiler is enabled.

By enabling the Map Debugger, you can do the following:

- Identify breakpoints.
- Step through rule execution.
- View object values at a rule level.

Breakpoints

You can set breakpoints on the output objects of functional and executable maps.

To add a breakpoint

You can only add a breakpoint when the Map Debugger is enabled. See "Running the Map Debugger" .

From an output card, right-click on an output object and select **Add BreakPoint**.

The output object rule with the breakpoint becomes highlighted.

To delete a breakpoint

Right-click on the output object and select **Delete BreakPoint**.

Saving breakpoints

When you save a map after adding breakpoints, the breakpoints are also saved. Therefore, the next time you open the map and enable the debugging option, the saved breakpoints will reappear. To remove the breakpoints, right-click on an object and select **Delete All BreakPoints**.

Conditional BreakPoints

You can configure the Map Debugger to stop on a specific iteration of a rule. For example, a rule repeats 99 times and the ninety-ninth occurrence is wrong. You can configure the Map Debugger to stop on the ninety-ninth iteration and subsequent iterations of the rule.

To set a conditional breakpoint:

1. Right-click on a card object and select **Show BreakPoints**.
The BreakPoints dialog is displayed.
2. Select an object from the list, and click **Condition**.
The BreakPoint Condition dialog is displayed.
3. In the **Condition** field, select **On Iteration**.
4. In the **Value** field, enter a numeric value that represents the iteration of the rule that you want the Map Debugger to begin stopping at. For example, enter 99 for the Map Debugger to begin stopping on the ninety-ninth iteration of the rule.
5. Click **OK**.

When the map runs in this case, the Map Debugger stops at the ninety-ninth iteration of the rule and then every occurrence of the rule thereafter.

Rule execution

Starting from the first breakpoint, the debugging process can step through every rule on every card. The debugging process can step into functional maps and follow RUN maps that are defined within the same map source file (provided that the maps have been built with the Debug option turned on).

If any referenced functional maps or external RUN maps are found, the map enters step-over mode and stops at the end of the originating rule.

Viewing object values at a rule level

When processing a rule, the debugger displays relevant data. Relevant data is data that is read and written in the rule. The data currently being processed by a rule or function is displayed in the Debug Rule window.

The rule bar in the Debug Rule window displays the current rule in a format of a hierarchical tree of objects, functions, and maps. At each breakpoint at which the map pauses, input and output values of objects, functions, and rules are shown.

Running the Map Debugger

When output from map execution is not what was expected, you can use the debug functionality to analyze each step of map execution.

To analyze map execution

1. From your map source file, select the map in question.
2. Choose **Map** → **Enable Debug**.
3. To set a breakpoint, right-click on an output object and choose **Add BreakPoint** from the context menu.

The object rule is highlighted in yellow or the color that you select (see Map Debugger configuration options).

4. Build and run the map.

Map execution pauses when it encounters the object for which the break was positioned.

The current rule is displayed in the Debug Rule window as shown:

5. At this point, you can choose from the following options:
 - **Step In** to execute the next object or function in the rule. For objects, the current value appears. For functions, the output displays the result of the particular function.
 - At a rule level, the debug process proceeds to the next function or returns the value of the next object in the current rule.
 - At the function level, the object or map evaluated is updated, displaying the data that was returned.
 - At the object level, the value of the object is displayed.
 - At the function and map level, the result of the particular operation is displayed.
 - **Step Thru** to step through the rule until you recognize the problem. When the current rule executes the results are displayed. At a function level this option does not debug the remainder of the functions/objects in the rule.
 - **Step Over** to move to the next rule in the current map.
 - **Continue** to continue map execution and stop at the next breakpoint if one exists, otherwise continue to the end of the map.
 - **Stop** to stop map execution. Execution will cease when all RUN maps have completed.

Remote debugging

You can also use the Map Debugger to debug maps that run remotely on the supported AIX, HP-UX, HP-Itanium, Linux, Solaris, Windows, and USS platforms.

Similar to using the Map Debugger to debug maps running locally on Windows, you can set breakpoints on rules, inspect data during the transformation process, and use the same commands.

To use this functionality, WebSphere Transformation Extender Design Studio (version 8.1 or later) is required on the local computer and a minimum of WebSphere Transformation Extender with Command Server (version 8.1 or later)

must be installed on the remote computer where the executable map will run. The map must be built using version 8.1 or later of the Map Designer and then deployed to the remote computer.

On the remote computer, you must first start the debugging server (for example, using Telnet) before you can begin the debug process from the Map Designer. When using Telnet, you can stop the debugging server by entering Ctrl+C in the Telnet command window.

Debugging a map remotely

These instructions assume that you have built a map using the Map Designer (version 8.1 or later) and have deployed the executable map file to the remote host.

Before you begin, access the remote host (for example, using Telnet) and set up your WebSphere Transformation Extender environment.

1. Start the debugging server by executing the applicable file (located in the product installation directory):
 - StartDebugServer.bat (Windows)
 - StartDebugServer.sh (UNIX, USS)

By default, the debugging server uses port 2590. To change the default port setting, you must modify the applicable batch or script file to include your specific port information. To stop the debugging server, press Ctrl+C.

2. On the local computer, open the map source file for the remote executable map to be debugged.
3. In the Navigator, select the specific map to debug.
4. Select **Map** → **Debug** to enable debug mode. (This also enables the Remote option.)
5. Add one or more breakpoints to the output. (From an output card, right-click on an output object and select **Add BreakPoint**.)
6. Select **Map** → **Remote**. The Remote Debugging window is displayed.
7. Under **Remote Host**, enter the remote computer information pertaining to the map to be debugged. In the **Host** field, enter the alias name or IP address of the remote computer. In the **Port** field, the default port number (2590) to be used by the debugging server is displayed.
8. Under **Platform**, select the platform of the remote host.
9. Under **Security**, enter a user name and password that provides access to the remote host where the executable map resides.
10. In the **Map** field, select the browse button and navigate to the executable map to debug.
11. Click **OK** to begin the remote debugging process. Similar to the regular map debugging process, a Debug window is displayed from where you can select various options (such as Step In, Step Thru, Step Out, Step Over, or Continue) after a break point is reached.

Trace files

The trace file is a text or XML file that records the steps of map execution. You can use the information in the trace file to diagnose invalid data or incorrect type definitions.

Trace for general data

The trace file is a text file that records map execution progress when enabled. You can enable the trace option from the map settings or from the adapter command line.

You can choose to trace the input data, the output data, or both. When input data is traced, the trace file provides a step-by-step account of the data objects found, the reason the data is found to be invalid, sizes and counts of data objects, and the data object position in the data stream. When output data is traced, the trace file specifies the objects that are built and the output objects that evaluate to "none".

Because performance can be impacted, enable the trace option only during debugging.

For input data tracing, a message exists in the trace file for each input data object indicating whether the data object is valid. If the data is invalid, the message indicates the reason the data is invalid.

Creating a trace file

The name and location of the trace file is defined in the Map Settings dialog box. The default name of the trace file is the executable map name, plus the file name extension .mtr. For example, if your executable map name is **MyMap**, the trace file is MyMap.mtr. The default location of the trace file is the map execution directory.

To create a trace file:

1. Select the map.
2. From the **Map** menu, choose **Settings**.
3. Specify the desired map trace settings.
4. Click **OK**.

The next time you execute the map, the trace file is created.

If you ran a map and got a message that an input was invalid, turn on the input trace only. If you got a message that an output was invalid, turn on the output trace only. If you trace both input and output, the input messages are at the beginning of the file, followed by the output messages.

Viewing the trace file

After you run a map, you can view the trace file on the **Trace** tab in the Organizer.

A trace file of input messages contains a message for each input data object. A message gives a date and time stamp, describes the level of the object in the card, the offset of the data object in the data stream, the length of the data object, its component number, the index of the component, a portion of the actual data, and the name of the type to which it presumably belongs.

In the trace file, the symbol `\r` indicates a carriage return. The symbol `\n` indicates a linefeed.

The DI number and X number are not used for debugging purposes and can be ignored.

For example, shown below is trace message for the **Name** data object:

A sample portion of a trace file:

```
18:22:17.837 Feb 13 2000
(Level 2: Offset 0, len 6, comp 1 of 2, #1, DI 00000001:)
Data at offset 0 ('Nicole') was found to be of TYPE
X'0004' (Name Data).
```

The level of an object indicates the position of the data object with respect to the entire card object. A card object has level 0; a component of the card object has level 1, and so on.

When data is valid

During data validation, a given data object is examined, and compared to its type definition. If a data object matches the type definition, the object is valid and the trace file indicates the object was found to be of that type. For example, in the trace message below, the data **Nicole** is a valid **Name** data object. It was found to be of the type **Name**.

```
18:22:17.837 Feb 13 2000
(Level 2: Offset 0, len 6, comp 1 of 2, #1, DI 00000001:)
Data at offset 0 ('Nicole') was found to be of TYPE
X'0004' (Name Data).
```

Group validation

A group is valid if the syntax is valid and all of its components are valid. A group is validated component-by-component. A component is valid if its type is valid, its component range is valid, and, if the component has a rule, the rule evaluates to "true".

When a component rule is evaluated, any invalid data used in a component rule is evaluated with a value of "none".

After all of the components of a group have been found, a message in the trace file indicates that the group itself was found.

For example, the components of **Row** are **Name** and **Phone#** and the data for **Row** is:

```
Nicole,352-2929
```

To find the object **Row** in the trace file, first the components of **Row** are found. The trace file contains a message for **Name**, then one for **Phone#**, and one for **Row**.

Valid card objects

If all of the objects that make up a card data object are found, the card object is valid. There is a trace message indicating this status. For example, if the data object for card 1 was valid, the trace message "INPUT 1 was valid" appears in the trace file. The message for a card object appears after all of the messages for the objects within that card.

For the Database object of an input card, the type **Database** is composed of **Row(s)**, and **Row** is composed of **Name** and **Phone#**.

The data for the entire input database is:

```
Nicole,352-2929
David,377-8098
```

In the trace file, there are messages for each **Row** and then there is a message for the entire card object, **Database**.

Invalid data

When a data object does *not* match the definition of the type to which it presumably belongs, it could be invalid.

Possible reasons for an invalid data object:

- Item value is not in restriction list
- Item failed the presentation test
- Item is the wrong size
- Data object has the wrong delimiter
- Data object has the wrong terminator
- Data object failed the component rule
- Group is missing a required component
- Group contains one or more invalid component(s)

Invalid card objects

If an entire card object is invalid, a runtime message appears in the Command Server window, indicating that the data is invalid. For example, if an input source is a file and the file is invalid, when the map is run, the message "One or more inputs was invalid" may appear.

If you trace the card containing the invalid object, a message in the trace file indicates that the card data object was invalid. If the data object of input card 1 is **File** and is invalid, the trace file would indicate this with the message "Input 1 exists, but it is invalid".

Prior to the message that the entire card object is invalid, there is an explanation of why it is invalid. For example, it might be that a required component of the card object is in error. The message about the invalid component will contain a negative word such as *not*, *failed*, *wrong*, or *invalid*.

Finding the important messages

The best way to read the trace file is open it in a word processor and use the search facilities. The following table explains the trace options to choose and what to look for in the trace file, based on the runtime message.

If You Get this Run-time Message:	Limit Trace To:	Read This in the Trace File:
One or more inputs was invalid.	The summary	Read the card summaries to find out which card is invalid.
	The invalid card	Start searching from the bottom, and look for the word "invalid" in the trace file.
Input valid, but unknown data found.	The summary	Read the card summaries to find out which card contains the unknown data.
	The card containing the unknown data	Start searching from the bottom, and look for the word invalid in the trace file.

Existence of data

When a certain data object is validated, determining the existence of that object occurs before fully validating all of the components within that object.

For example, if a data object is a partition and includes an initiator, as soon as the initiator is found, it is known that the data object exists. A message exists in the trace file, indicating that the data object is "known to exist".

The existence of a certain object can be determined if that object has certain properties. For a detailed explanation of how the existence of an object is detected, see the Type Designer documentation.

Example of existence of record objects

The following example shows how the existence of record objects can be determined. An input file is composed of records with three kinds of records: **A**, **B**, and **C**.

Record is partitioned into **A**, **B**, and **C**, the three types of records. Each record begins with a letter followed by an asterisk. The letter indicates the kind of record. This value is defined as the initiator for each record type. For example, an **A** Record begins with **A***.

Shown below is the data:

A*couch,1

B*plate,12

A*chair,6

C*sweater,15

B*glass,8

Because the initiator partitions **Record**, as soon as the initiator of a given record is found, that record is known to exist.

Validation for invalid objects

If an object is determined to be invalid, and that object is known to exist, validation continues. For example, there was an error in the first **B Record**.

It is known that **B Record** exists. When **B Record** is found to be invalid, validation continues with the following records.

If an object is *not* known to exist, the object is found as unknown data.

Unknown data

When input data is validated, additional data that does not validate as part of the input may also be found. This data is unknown data.

An example input is composed of **Product(s)** and **Product** is *not* defined with properties that allow its existence to be predetermined.

The second **Product** is invalid because the **Quantity** field is missing.

When the invalid **Product** is found, it is not known to exist. Therefore, it is assumed to be unknown data.

Sometimes a message about unknown data is not significant. You may get a message about unknown data because there are extra carriage returns or "junk" data at the end of an input file. You should read the trace file to determine whether all of the pertinent input data was found.

Negative messages not fatal

Sometimes validation of a given data object interprets the given data object as belonging to a type to which it does not really belong. In this situation, the data object will fail to belong to that type - but that is because the data does not belong.

In the following two situations, you may see messages that a data object fails to belong to a certain type because it is not really data of that type:

- When an optional occurrence of a component that does not exist in the data is sought during validation
- When an object of a partitioned type is sought during validation

Example of optional occurrences

If an occurrence of a component is optional, there may be more than one possibility for a given data object. For example, suppose a type has the following components:

A (1:3)

B

After **A** is found, the next object in the data stream can either be **A** or **B**. Suppose it is **B**. Therefore, the object fails to be **A**. A message in the trace file indicates that the object does not match the definition of **A** in some way. After the validation process has determined that the object is not **A**, validation occurs to determine whether it is **B**. Then the object is validated as **B**.

It is acceptable if an *optional* occurrence of a component does not exist in the data; it is optional. However, if a *required* occurrence of a component does not exist, the data object containing that component is invalid.

The optional **Message** is missing from the data. After finding the **Header**, the next data object searched for is **Message**. However, the next object in the data is not a **Message**, it is a **LineItem**.

The trace file indicates that the initiator of the data object is not **M** which is the initiator of a **Message**. After it is determined that the data object is not a **Message**, the **LineItem** is validated as a **LineItem**.

If you see a message in the trace file saying that a data object does not exist, this is not necessarily the error causing the entire input object to be invalid. Keep in mind the data being validated, and the optional data. Your understanding of optional data is important when reading the trace file.

Example of partitioned types

Another case in which a data object might fail to belong to a certain type is when the type is partitioned. If an object that is presumably a partition is validated, the validation process goes down the list of partitions to determine the partition to which the object belongs. You might see a message that indicates an object failed to be a certain partition. That means it does not belong to *that* partition, it may

belong to one of the other partitions. The validation process checks to see whether it belongs to the next partition, and so on.

For example, suppose **Record** is partitioned into **A**, **B**, and **C**.

Each record begins with an initiator: **A***, **B*** or **C***.

First, validation checks the initiator to determine whether it is an **A Record**. The initiator will be the wrong value for an **A Record**. Next, validation checks the initiator to determine whether it is a **B Record**. The initiator will be the wrong value for a **B Record**. Next, validation checks to determine whether it is a **C Record** and the object is validated as a **C Record**.

Viewing results in hex mode

Run results can be viewed in hex mode, which displays the offset values of the data. This may be helpful if the trace file indicates a problem at a particular offset in the data. To view run results in hex mode, enable the **Display In Hex Mode** check box in the **Run Results** dialog box.

Trace for XML data

An XML-formatted trace file that contains XML validation information can be created during map execution for maps that contain XML type trees. When enabled, the XML trace file (*mapname_log.xml*) is generated in the map directory in addition to the general trace file. The XML trace file is not generated when no input card is present in the map.

For maps that contain XML type trees (created using Xerces), the input XML validation information that is generated to the regular trace file (.mtr) or that is viewable on the **Trace** tab of the Organizer is not valid. See the XML trace file (*mapname_log.xml*) for accurate XML input validation information.

When viewed with Microsoft Internet Explorer, the XML trace file:

- Contains a validation log.
- Displays the XML validation sequence.
- Presents validation results.

When using a browser other than Microsoft Internet Explorer, only the content of the XML file is displayed.

Similar to the general trace, you must enable certain **MapTrace** settings to generate an XML trace file.

To generate an XML trace file

1. In the Map Designer, right-click the map that contains references to XML type tree(s) and choose **Settings**.
2. Set the **Map Trace Content Trace** → **Switch** setting to **ON**.
3. Set the **Map Trace Content Trace** → **Input ContentTrace** setting to **All**.

After running the map, you can find the XML trace file (*mapname_log.xml*) in the map directory (unless otherwise specified). The file opens in your default browser. When you run the map again, the existing file is overwritten.

XML validation log

The **Source** tab displays the XML validation log. The XML validation log is a step-by-step representation of the XML validation process. This log presents the time of map execution, input file information, and other validation details.

Validation sequence

The **Validation Sequence** tab displays the reverted order in which the XML validation library identified the types from the XML input file.

The XML validation sequence can be useful in troubleshooting, for example, to see where (or on which type) the validation process stopped.

Validation results

The **Validation Result** tab displays the result of the XML validation process. When XML validation does not complete successfully, the applicable error messages are displayed. If map execution failed early in the validation process, this tab might not contain any information.

To locate the error, open the XML input file, go to the fourteenth line, and then 10 characters into the line.

The following image displays the XML input file for this example. The validation error is associated with **GenDate**. (**GenDate** is required and there is no value present.)

Chapter 21. Error and warning messages

Map execution error and warning messages

After a map is executed, map execution messages indicate the execution results. If audit is enabled, these messages are also recorded in the appropriate audit log for the map.

The following table lists all the codes and messages that can be returned as a result of running a map.

The return codes marked with an asterisk (*) are warning codes. See "Map Execution Warning Messages" for a consecutive list of the warning codes for a map.

Return Code	Message
0	Map completed successfully
1	User aborted map If you cancel the Server before the map completes, this message is displayed.
2	Not enough memory to execute map There is not enough memory to initialize the map. Try closing all other running applications.
3	Could not open map The .mmc file cannot be found. Ensure the file name or path is correct.
4	Could not read map If you have ported your map and sent it with FTP to another platform, make sure you have done so using the BINARY option so the map's contents will not be corrupted.
5	Could not read inputs
6	Invalid map handle
7	Invalid card number was specified Your map may be corrupt.
8	One or more inputs was invalid Turn the input trace on, run the map again and read the trace file.
9	Target not available An incorrect file name or path for a target may be specified.
10	Internal error
11	Could not build one or more outputs An attempt may have been made to write to a read-only directory, or an output file directory may be missing.

- 12 Source not available
An incorrect file name or command for a source may be specified.
- 13 Could not open work files
An invalid path for work files may be specified.
- 14* One or more outputs was invalid
A number produced an overflow condition. Enable the **Trace Output** option, run the map again, and read the trace file.
- 15 Map must be recompiled
You may be trying to run a map with a server that has a different version from the Map Designer version used to compile the map. Make sure the Map Designer and the Server versions match.
- 16 Disk write error
An attempt may have been made to write to a read-only directory or file. Also, check the amount of available disk space.
- 17 Disk read error
An attempt may have been made to access a file on a shared resource that is not accessible. Also, check the amount of available disk space.
- 18* Page usage count error
Your data mapped correctly; however, an internal paging problem exists.
- 19 Internal calling error
- 20 Reopen file failed
Your map did not run correctly.
- 21* Input valid but unknown data found
Enough of the input data to conform to the card definition was recognized, but there was more data at the end of the input stream. Enable the **Trace Input** setting, re-run the map and read the map trace file (.mtr).
- 22 Page size too small
Increase the page size or decrease the number of types in the type tree that are used to define the input.
- 23 Unable to reuse work file
Use the same page settings that were used when you created the existing work file.
- 24 Database error
A problem was encountered attempting to write to an output database. Read the database trace file (.dbl) for information about possible causes.
- 25 File attribute error
The Command Server or Launcher failed to write data to either the work or data file. Check the output card definition on the map to see if it matches the platform-specific data definition specified for the file the map is attempting to create.
- 26* Output type in error
The entire card object is in error.

- 27* Output type contains errors
One or more components failed restrictions or a component rule.
- 28* Input type contains errors
This occurs when mapping data of a component that has a RESTART assigned to it, and at least one instance of the component in the input data is invalid.
- 29* Output valid, but unknown data found
This occurs when enough of the output data is recognized to conform to the card definition, but there was more data at the end of the output stream.
- 30 FAIL function aborted map
This occurs when the FAIL function is used within the map to cause map execution to cease. The return message specified as the second argument of the FAIL function appears in place of the *xxx* in the Execution Audit section of the Audit Log. See the Functions and Expressions documentation for more information about using the FAIL function.
- 31 This map requires DSTX for Japan
This map uses Japanese characters and must be executed using WebSphere Transformation Extender for Japan.
- 32 Invalid map instance handle
- 33 Map instance handle in use
- 34* Too few pages requested, more allocated
The map can complete but it can be better optimized by increasing the page count.
- 35 Insufficient number of pages to execute map
The map could not complete because not enough pages were specified.
- 36 Could not open work files
A scratch file for the HANDLEIN function could not be created.
- 50 Not enough memory to execute map
There is not enough memory to initialize the map. Try closing all open applications.
- 51 Card override failure
There is a problem with the command line entered for an -I or -O option.
(Applicable to Command Server)
- 52 I/O initialization failure
A run map was called and could not allocate memory for input or output.
- 53 Open audit failure
The audit file was not accessible.
- 54 No command line
There was no map or command line specified.
(Applicable to run maps and the Command Server)

- 55 Recursive command files
A command file was specified within a command file.
(Applicable to run maps and the Command Server)
- 56 Invalid command line option - <option>
An invalid argument was entered on the command line.
(Applicable to run maps and the Command Server)
- 57 Invalid `W` command line option
An invalid parameter for the -W option was specified.
(Applicable to run maps and the Command Server)
- 58 Invalid `B` command line option
An invalid parameter for the -B option was specified.
(Applicable to run maps and the Command Server)
- 59 Invalid `R` command line option
An invalid parameter for the -R option was specified.
(Applicable to run maps and the Command Server)
- 60 Invalid `A` command line option
An invalid parameter for the -A option was specified.
(Applicable to run maps and the Command Server)
- 61 Invalid `P` command line option
An invalid parameter for the -P option was specified.
(Applicable to run maps and the Command Server)
- 62 Invalid `Y` command line option
An invalid parameter for the -Y option was specified.
(Applicable to run maps and the Command Server)
- 63 Invalid `T` command line option
An invalid parameter for the -T option was specified.
(Applicable to run maps and the Command Server)
- 64 Invalid `G` command line option
An invalid parameter for the -G option was specified.
(Applicable to run maps and the Command Server)
- 65 Invalid `I` command line option for input <2>
An invalid parameter for the -I option was specified for
input card <number>.
(Applicable to run maps and the Command Server)
- 66 Invalid size in echo command line for input < 2 >
A size (nnn) was specified for -IE2Snnn that is outside of the boundaries
for input card <number>.
(Applicable to run maps and the Command Server)

- 67 Invalid adapter type in command line for input <2>
Using the -IAadapter_name or -IMadapter_name, <adapter_name> is not a valid adapter type.
(Applicable to run maps and the Command Server)
- 68 Invalid `O` command line option for output <1>
An invalid parameter for the -O option was specified for output card <number>.
(Applicable to run maps and the Command Server)
- 69 Invalid adapter type in command line for output <1>
Using the -OAadapter_name or -OMadapter_name, adapter_name is not a valid adapter type.
(Applicable to run maps and the Command Server)
- 70 Command line memory failure
Memory allocation failure during processing of command line.
(Applicable to run maps and the Command Server)
- 71 Invalid `D` command line option
An invalid parameter for the -D option was specified.
(Applicable to run maps and the Command Server)
- 72 Invalid `F` command line option
An invalid parameter for the -F option was specified.
(Applicable to run maps and the Command Server)
- 73 Resource manager failure
The resource manager could not reserve the resources (sources/targets) due to a resource contention that is most likely a run map.
(Applicable to run maps and theLauncher)
- 74 Invalid `Z` command line option
An invalid parameter for the -Z option was specified.
(Applicable to run maps and the Command Server)
- 75 Adapter failed to get data on input
During the commit/rollback phase, the source had a failure
- 76 Adapter failed to put data on output
During the commit/rollback phase, the target had a failure.
- 77 Invalid map name: <map_name>
The map path and name exceed 260 bytes.
(Applicable to run maps and the Command Server)

Map execution warning messages

The following table lists the return codes that are warnings only - meaning that the map did complete, but some problems occurred. You can change the default behavior for the warning codes by specifying **Fail on Warnings** or **Ignore Warnings** through the map settings or by using the -F or -Z execution command.

The warning codes marked with an asterisk (*) are returned only when at least one output is listed in the **Data Audit Settings** of the **Organizer** and in the **Map Settings** dialog, **MapAudit** is enabled with **Data** enabled for the **Log** setting.

Return Code

Message

- | | |
|-----|---|
| 14 | One or more outputs was invalid
A number produced an overflow condition. Enable the Trace Output option, run the map again, and read the trace file. |
| 18 | Page usage count error
Your data mapped correctly; however, an internal paging problem exists. |
| 21 | Input valid but unknown data found
Enough of the input data to conform to the card definition was recognized, but there was more data at the end of the input stream. Enable the Trace Input setting, run the map again and read the map trace file (.log). |
| 26* | Output type in error
The entire card object is in error. |
| 27* | Output type contains errors
One or more components failed restrictions or a component rule. |
| 28 | Input type contains errors
This occurs when mapping data of a component that has a RESTART assigned to it, and at least one instance of the component in the input data is invalid. |
| 29* | Output valid, but unknown data found
This occurs when enough of the output data is recognized to conform to the card definition, but there was more data at the end of the output stream. |
| 34 | Too few pages requested, more allocated
The map can complete but it can be better optimized by increasing the page count. |

Map build error messages

Build errors prevent a map from being compiled. If build errors occur, view the errors, correct the problem(s), and build the map again. A map cannot be run if it has not been built.

In the following build messages; these letters are used as variables:

x = the map where the error occurred

y = the output that contains the rule in error

w, z = numbers

Words used as variables are in *italics*. For example, *map-name* represents the name of a map.

Return Code

Message

M100 Analysis aborted by user.

M101 Map: x

Unresolved rules exist.

Hint: Go to map x . If you want to use the unresolved rules, copy and paste them. Delete any unwanted unresolved rules.

M102 Map: x Output: y

Functional map referenced in rule must have one (and only one) output.

Map referenced: *map-name*.

Hint: Go to the functional map. Delete the extra card(s).

M103 Map: x Output: y

Number of arguments for functional map *map-name* is incorrect.

Number specified: *#specified* Number required: *#required*

Hint: Add or delete arguments from the rule, or add or delete input cards in the functional map.

M104 Map: x Output: y

Empty rule cell. Rule required.

Hint: Go to map x . Enter rules in each empty rule cell.

M105 Map: x Output: y

Rule references unknown type: *type name*.

Hint: You may have made a change in the type tree. If the card object is unknown, edit the card, and choose a valid type. You may have incorrectly typed in the name. If you drag and drop the type, the correct name is automatically entered.

M106 Map: x

Circular map reference from map *map-name*.

Analysis must terminate.

Hint: Two different maps contain rules that reference each other. Remove the reference from one of the maps.

M107 Map: x Output: y

Rule syntax is invalid starting at character position: *position#*.

Rule

Hint: This error may occur if a previous version did not catch the syntax error. Generally, syntax errors are caught as rules are entered. see the Release Notes for any changes in rule syntax.

- M108** Map: x Output: y
 Rule references unknown map: *map-name*.
 Hint: Re-enter the map name in the rule. You may have typed the map name incorrectly. Map names are case-sensitive. You may have forgotten to create the functional map. If so, create it.
- M109** Map: x Card: *cardname*
 Card is invalid.
 Hint: The type name is either misspelled or you may have made a change in the type tree by deleting or renaming the type. Edit the card, and choose a valid type.
- M110** Map: x Card: *cardname*
 Card type is not a group or item.
 Hint: You probably changed your type tree. Edit the card, and choose a group or item.
- M111** Map: x Output: y
 Argument $\#w$ for map *map-name* does not match type of input card $\#z$.
 Hint: Change argument $\#w$ so that it matches the type of input card $\#z$, or change the type of input card $\#z$ so that it matches argument $\#w$.
- M112** Map: x Output: y
 Argument $\#w$ for map *map-name* does not match item sub-class of card $\#z$.
 Hint: Change argument $\#w$ so that it matches the item sub-class of input card $\#z$, or change the type of input card $\#z$ so that its item sub-class matches argument $\#w$.
- M113** Map: x Output: y
 Argument $\#w$ for function *function name* is invalid.
 Hint: Select the rule for output, y . Right-click the map rule and choose **Properties** from the context menu. The **Rule Properties** dialog box allows you to view function arguments and object properties. You have either referenced an invalid object, or used the wrong type of argument.
- M114** Map: x Output: y
 Output argument of rule does not match output type: *rule*.
 Hint: The map rule is resulting in a data object of the wrong type. Check the functions and the maps used within the map rule and make sure that the entire rule evaluates to the correct type.
 You may have dragged and dropped a group that is a different type to this output. Use a functional map instead. If the rule contains the name of a functional map, go to the functional map and edit the output card. Make sure the type of the output card is the output y .
- M115** Map: x Output: y
 Output argument of rule does not match output item sub-class: *rule*.
 Hint: Notice the output y . The output argument of the rule must have the same item sub-class as y .
- M116** Map: x Output: y

Type Tree for this card is not used in executable map.

Hint: Edit the card and select a type tree that is used in the executable map.

M117 Map: x Output: y

Cannot assign an output to itself.

Hint: You might have accidentally dragged and dropped an output into its own rule cell. Edit the rule.

M118 Map: x Output: y

Rule references unknown comment: *commentname @ inputname*

Hint: The input card type does not have a floating component, or you have incorrectly typed in the name. If the input card has a floating component, on the **Card** menu, the **Show Floating Component Type** option will be activated.

M119 Map: x Output: y

Output argument of rule is not a partition of output type.

Hint: If the rule contains the name of a functional map, go to the functional map and edit the output card. Make sure the type of the output card is the output y , or a partition of the output y .

M120 Map: x Output: y

Operand is an invalid operand.

Hint: For information on operands, see the Functions and Expressions documentation.

M121 Map: x

Input card *#cardnumber* uses invalid tree.

Hint: You may have moved the type tree or map source from its original location. You may have incorrectly entered the name of the type tree.

M122 Map: x

Output card *#cardnumber* uses invalid tree.

Hint: You may have moved the type tree or map source from its original location. You may have incorrectly entered the name of the type tree.

M123 Map: x Card: *cardname*

Card updates input, but card type and data file do not match an input.

Hint: Edit the card and re-define the update information.

M124 Map: x Output: y

Rule references type later in output: *rule*

Hint: If possible, enter an equivalent rule in a later output. An output referenced in a rule must appear above the rule within the output card.

M125 Map: x

Invalid audit: *audit statement*

Hint: Look at the Audit Settings window. An asterisk * appears next to any invalid audit statement. Re-enter the audit statement.

Map build warning messages

A build warning alerts you of a condition that might be problematic. If you build a map and there are warnings, but no errors, the map is compiled. You can ignore warnings. To ignore warnings, from the **Tools** menu, choose **Options**, and select **General**. Enable the **Suppress build warnings** check box.

Return Code

Message

M200 Map: x Output: y
Size of input item is greater than size of output item.

Map compile error messages

The compile errors display on the **Build Results** tab of the **Organizer**. Maps with compile errors may not be run. Correct all compile errors to run a map.

The following table lists the errors than can result when a map is built:

Return Code

Message

- M1001** Compile aborted by user.
Hint: You cancelled the build process before it completed.
- M1002** Not enough memory available to compile.
Hint: Close other applications that you have open, and build the map again.
- M1003** Couldn't open the map source file.
Hint: The map may be corrupt. Please contact Technical Support.
- M1004** Couldn't find the executable map.
Hint: The map source file may be corrupt. Please contact Technical Support.
- M1005** Duplicate map names in output file.
Hint: If you get this error, please contact Technical Support.
- M1006** Tree used in functional map not used by executable map.
Hint: Edit the card in the functional map or the executable map so both cards reference the same tree.
- M1007** No delimiter defined for delimited type.
Hint: In the Type Designer, specify the delimiter value for the given type.
- M1008** Unable to open the type tree for input card.
Hint: The type tree may not exist in the directory specified, or the type tree may be corrupt. If your type tree is corrupt, please contact Technical Support.
- M1009** Unable to open the type tree for output card.
Hint: The type tree may not exist in the directory specified or the type tree may be corrupt.

- M1010** Unable to find a type in an input tree.
Hint: If you get this error, please contact Technical Support.
- M1011** Unable to find a type in an output tree.
Hint: If you get this error, please contact Technical Support.
- M1012** Could not retrieve query information.
Hint: Either you have not installed the Database Interface Designer or the .mdq file is incorrect.
- M1013** Could not retrieve database information.
Hint: Either you have not installed the Database Interface Designer or the .mdq file is incorrect.
- M1014** Input data file for executable map missing or invalid.
Hint: Check the file names used for sources and targets.
- M1015** Output data file for executable map missing or invalid.
Hint: Check the file names used for sources and targets.
- M1016** Parser found an invalid rule.
Hint: If you get this error, please contact Technical Support.
- M1017** Output card referenced in rule not found.
Hint: If you get this error, please contact Technical Support.
- M1018** Invalid component or partition subtype encountered.
Hint: If you get this error, please contact Technical Support.
- M1019** Unable to resolve type references in rule.
Hint: If you get this error, please contact Technical Support.
- M1020** Couldn't open the compiled map file.
Hint: The compiled map file may be corrupt. Please contact Technical Support.
- M1021** Error writing to the compiled map file. (Disk full?)
Hint: Delete unnecessary files from your hard disk.
- M1022** Unable to resolve type references in audit.
Hint: If you get this error, please contact Technical Support.
- M1023** Unable to open .MDQ file.
Hint: The .mdq file does not exist or is otherwise inaccessible.
- M1024** Database not found.
Hint: The named database is not defined in the specified .mdq file.
- M1025** Database Error.
Hint: An unexpected error occurred while processing and .mdq file.
- M1026** Input MDQ file not found.
Hint: The .mdq file does not exist or is otherwise inaccessible.
- M1027** Input query not found.

Hint: The named query is not defined in the specific .mdq file.

M1028 Output MDQ file not found.

Hint: The .mdq file does not exist or is otherwise inaccessible.

M1029 Output database not found.

Hint: The named database is not defined in the specified .mdq file.

M1030 At least one syntax object literal is too long.

Hint: Type Designer analysis is required.

Audit log status codes

The status attribute in the Audit Log is a letter followed by a code number. The letter of the status attribute tells whether the data is valid (V), or caused an error (E), or caused a warning (W).

Valid audit log status codes

The following table explains the number in the status code if the letter is V.

Status Code	Description
V00	The object and all contained objects are valid
V01	The object is valid, but it contains invalid object(s)

Warning audit log status codes

The following table explains the number in the status code if the letter is W.

Status Code	Description
W01	Object failed restriction
W02	Object failed presentation
W03	Object failed size check

Error audit log status codes

The following table explains the number in the status code if the letter is E.

Status Code	Description
E00	Object is an unidentified foreign object (UFO) - data not associated with any particular type
E01	Object failed restriction
E02	Object failed presentation
E03	Object failed size check
E04	Invalid or missing initiator
E05	Invalid or missing terminator
E06	Object missing required component

- E07 Object is invalid because it contains components in error
- E08 Object failed partitioning
- E09 Object failed component rule
- E10 This required object is missing
- E11 Invalid or missing delimiter
- E12 More instances of an object exist than have been specified (Only implemented for unordered groups)
- E13 Object failed to meet minimum size requirement.
- E14 Object failed to meet maximum size requirement.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both:

AIX
AIX 5L
AS/400
Ascential
Ascential DataStage
Ascential Enterprise Integration Suite
Ascential QualityStage
Ascential RTI
Ascential Software
Ascential
CICS
DataStage
DB2
DB2 Universal Database
developerWorks
Footprint
Hiperspace
IBM
the IBM logo
ibm.com
IMS
Informix
Lotus
Lotus Notes
MQSeries
MVS
OS/390
OS/400
Passport Advantage
Redbooks
RISC System/6000
Roma
S/390
System z
Trading Partner
Tivoli

WebSphere
z/Architecture
z/OS
zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

This product includes software developed by the Eclipse Project (<http://www.eclipse.org/>).



IBM WebSphere Transformation Extender, Version 8.1

Index

Special characters

\$ in a map rule 92

A

arguments

- of a functional map
 - deciding on 81
 - syntax 80

audit log 145

- a particular data object 151
- auditing a map 150
- data content 149
- error status codes 178
- mapping from audit log 151
- using to track invalid data 156
- valid status codes 178
- warning status codes 178

automatic item conversions 71

B

Backup card settings 52

- defining 52

breakpoints

- conditional 157
- deleting 157
- saving 157
- setting 156

build results 8

- viewing 131, 136

build warnings 131

building a map 132

- after modifying 126
- analyzing a type tree before 130
- when to build 127

bursts 111, 113

- batching logical messages 111
- configuring for error recovery 112
- configuring the RUN function and bursts 113
- in the audit log 147, 148
- using for large data 111
- using for large files 111

C

Card menu 14

Card menu commands

- Convert to Input/Output 15
- Copy 15
- Delete 14
- Edit 14
- New 14
- Reorder 15

card settings 39

- CardName 39
- differences 32
- exporting 119
- input and output cards 39

card settings (*continued*)

- TypeName 40

CardName 39

- card setting 39
- guidelines for naming 39

cards 36

- compositional hierarchy 36
- converting 123
- copying 123
- deleting 124
- editing 124
- managing 123
- moving 123
- multiple in a functional map 84
- name changing 124
- names in map rules 62
- naming guidelines 39
- specific details 35

Century map settings 105

choice groups 7

color coding map rules 55

command access 11

Command card setting 44, 48

command server

- run results displayed in 135
- using to run a map 135

comments in map rules 61

comparing

- map source files 30

component names in map rules 62

components

- viewing 65

composition view

- Navigator window 4

compositional hierarchy

- of cards 36

concatenating text in a map rule 72

CONTAINSERRORS function 144

converting

- cards 123

copy command 123

copying

- a map 116
- cards 123

creating

- map in a map source file 30
- map source file 29

customization options 18

D

data

- errors

 - tracking 156

- evaluating to NONE 69

- incrementing using a functional map 92

- invalid

 - tracking 156

- missing 69

- reordering 92

data audit 150

- data audit (*continued*)
 - configuring 151
 - elements in 151
- data audit settings 129
 - differences 32
- data errors
 - recovering from 143
- data object name
 - in map rules 61
- DatabaseQueryFile card setting 44
- debugging
 - a map 155, 156, 158
 - data errors
 - using the audit log 156
 - runtime errors 162
- deleting
 - a map 117
 - cards 124
- DocumentVerification setting 51
- drag procedures 115

E

- Edit menu commands
 - Copy 12
 - Cut 12
 - Find 12
 - more find/replace 13
 - Paste 12
 - Replace 13
 - Undo 12
- editing
 - cards 124
- editing a type tree
 - effects on a map 126
- ellipses 63
- error audit log status codes 178
- errors
 - data 143
 - tracking 156
- evaluation
 - of map rules 58
- example
 - mapping from audit log 151
- executable map
 - definition of 79
- expression
 - examples 56
- EXTRACT function 69, 80

F

- FetchAs card setting 41
- FetchUnit
 - logical or adapter-specified 113
- FetchUnit input card setting 42
 - FetchUnit details 43
 - with multiple input cards 43
- File menu 11
- File menu commands
 - Close 11
 - Exit 12
 - Map Source File Differences 12
 - New 11
 - Open
 - File 11

- File menu commands (*continued*)
 - Print 12
 - Print Setup 12
 - recently used files 12
 - Save 11
 - Save As 11
 - Source Control 11
- file name extensions 17
- FilePath card setting 44, 48
- files
 - recently used (File menu) 12
- floating component type
 - mapping from 74
 - viewing 74
- From window 5
- functional map
 - arguments
 - multiple 84
 - inputs
 - using multiple 84
 - name
 - in a map rule 80
 - syntax of 80
 - using to increment data 92
 - using to re-order data 92
 - when data is missing 81
- Functional Map Wizard 87
 - editing card definitions 89
 - referencing a map from 88
- functional maps 78
 - creating 87, 90
 - multiple levels 91
 - referencing 120
- functions
 - CONTAINSERRORS 144
 - EXTRACT 69, 80
 - INDEX 80
 - inserting into map rules 59
 - ISERROR 143
 - REJECT 143

G

- GET card setting 44
- group
 - map rule for 72
 - validation 161
- guidelines
 - for map names 30
 - for naming map cards 39
 - for reusing work areas 42

H

- Help menu 16
- Help menu commands
 - About Map Designer 16
 - Contents 16

I

- icons
 - cards
 - input and output 35
 - Navigator window 3
- incrementing data 92

- index
 - map rules 63
 - removing 73
- INDEX function 80
- indexing
 - an input 74
- input
 - of a functional map
 - deciding on 81
 - multiple 84
- input and output card settings 39
- input card 35
 - definition of 35
 - specific details 35
- Interval card setting 50
- invalid data
 - recovering from 143
 - tracking
 - with the audit log 156
- ISERROR function 143
- item
 - automatic item conversions 71
 - map rule for 71

L

- list view
 - Navigator window 4

M

- managing
 - cards 123
- map
 - analyzing a type tree before building 130
 - building after modifying 126
 - executable
 - definition of 79
 - functional
 - arguments 84
 - in a map rule 80
 - inputs 84
 - syntax of 80
 - modifying
 - effects on map rules 125
 - referencing 121
 - renaming 118
 - rule
 - after modifying a map 125
 - for a group 72
 - for an item 71
 - for an output with a range 72
- map build
 - errors 172
 - warnings 176
- map compile errors 176
- Map Debugger
 - overview 156
 - remote debugging 158
 - using 158
- Map Designer
 - input and output cards 35
 - using 1
- map execution
 - errors and warnings 167
 - warnings 172

- Map menu 13
- Map menu commands
 - Build 14
 - Build for Specific Platform 14
 - Close All Run Results 14
 - Copy 13
 - Copy Data Audit Settings 14
 - Delete 13
 - Merge 13
 - New 13
 - Organizer 14
 - Reference 13
 - Rename 14
 - Run 14
 - Run Results 14
 - Where used 14
- Map Profiler
 - configuring 139
 - overview 139
 - tips for best results 142
 - using 140
- map rule
 - after modifying a map 125
 - definition of 56
 - evaluation 58
 - for a group 72
 - for a group with a range 73
 - for an item 71
 - for an output with a range 72
 - referring to an output in 92
- map rules 55
 - color coding 55
 - comments 61
 - data object name in 61
 - entering 59
 - exporting 120
 - partition in 63
 - unavailable (grayed) rule cells 58
 - unresolved map rules 128
 - viewing 8
- map settings 95
 - defining default 95
 - defining for a single map 95
 - differences 32
 - exporting 119
 - printing 26
- map source file
 - creating 29
 - differences 30
 - referencing 121
- MapAudit map settings 96
- maps
 - analyzing performance 139
 - audit log status codes 178
 - building 127
 - copying 116
 - definition of 29
 - deleting 117
 - error audit log status codes 178
 - functional 77, 87
 - map build errors 172
 - map build warnings 176
 - map compile errors 176
 - map execution errors and warnings 167
 - map execution warnings 172
 - merging 117
 - messages 167

- maps (*continued*)
 - moving 116
 - name guidelines 30
 - porting 132
 - valid audit log status codes 178
 - warning audit log status codes 178
- MapTrace settings 101
- MaxAttempts card setting 50
- menus
 - Card 14
 - Edit 12
 - File 11
 - Help 16
 - Map 13
 - Rules 15
 - Tools 15
 - View 13
 - Window 16
- merge command 117
- merging maps 117
- missing data 69
 - when an input is missing 69
 - when an output is missing 69
 - when using a functional map 81
- mouse shortcuts 116
- moving
 - a map 116
 - cards 123

N

- Navigator 3
- Navigator window
 - composition view 4
 - docking display option 5
 - icons 3
 - list view 4
- NONE
 - example of using 69
 - inserting into a map rule 69
 - when an input argument of a functional map evaluates to 81
 - when an output evaluates to 69
 - when data evaluates to 69
 - when to use in a map rule 69

O

- object name
 - in map rules 61
- OnFailure card setting 49
- OnSuccess card setting 48
- Options dialog box 18
- Organizer window 7
- output
 - referring to in a map rule 92
 - with a range
 - map rule for 72
- output card 35
 - definition of 35
 - specific details 35
- output data
 - building 1

P

- partitions in map rules 63
- ported map
 - transferring 134
- porting a map 132
- printing
 - map details 25
- PUT card setting 48

R

- referenced maps
 - and using a resource alias 121
- REJECT function 143
- remarks
 - differences 32
- Remarks tab 129
- remote debugging 158
- renaming a map 118
- reordering
 - indexed objects 73
- resource aliases 121
- restart attribute
 - using 143
- Retry card setting 50
- rule
 - map
 - after modifying a map 125
 - for a group 72
 - for an item 71
 - for an output with a range 72
- Rule Bar 8
 - display options 9
 - showing and hiding 10
- Rules menu 15
- Rules menu commands
 - Delete 15
 - Functional Map Wizard 15
 - Goto Functional Map 15
 - Insert Function 15
 - Insert NONE if Empty 15
 - Insert Symbol 15
 - Properties 15
 - Show All 15
- run options 135
- run results
 - display options 136
 - printing 27
 - shortcut access method 137
 - viewing 136

S

- Scope card setting 46, 49
 - FetchAs and Scope actions 46
- searching
 - in the organizer window 131
- selecting
 - TypeName for card 40
- selecting objects 115
- selection capability 115
- single-sourcing maps 121
- size requirements for a type 97
- SizeValidation 97
- Source card setting 44
- SourceRule settings 40

- SourceRule settings *(continued)*
 - overrides 40
- special characters
 - in map rules 60
- status codes for audit log
 - error 178
 - valid 178
 - warning 178
- status codes in the audit log 153
- symbol
 - inserting into map rules 59
- syntax
 - of a functional map 80
- syntax errors in map rules 61
- syntax object
 - automatic conversion 72

T

- TargetRule card settings 47
- TargetRule settings
 - overrides 47
- text
 - concatenating in a map rule 72
 - entering in a map rule 72
- theory of operation
 - Map Designer 1
- To window 5
 - shortcut menus 6
 - viewing choice groups 7
- toolbar 10
- Tools menu 15
- Tools menu commands
 - Options 16
 - Short Cuts 16
- trace file 159
 - viewing 129
- trace files 103
 - ranges for InputContentTrace 103
- Transaction card setting 45, 48
- type
 - editing a type referenced in a card 126
- type properties
 - viewing 64
- type tree
 - analyzing 130
- TypeName
 - card setting 40
- TypeTree card setting 39

U

- unresolved map rules
 - differences 32
- Unresolved Rules tab 128

V

- valid audit log status codes 178
- validation map settings 106
- View menu 13
- View menu commands
 - Navigator 13
 - Rule Bar 13
 - Status Bar 13
 - Toolbar(s) 13

- viewing
 - build results 131
 - floating component type 74

W

- warning audit log status codes 178
- warnings
 - build 131
- Warnings settings 108
- Window menu 16
- Window menu commands
 - Arrange Icons 16
 - Cascade 16
 - Close All 16
 - From 16
 - Tile Horizontally 16
 - Tile Vertically 16
 - To 16
- wizard
 - for functional maps 87
- WorkArea card setting 42
 - guidelines for using 42
- WorkSpace settings 103

X

- XML
 - deploying to non-Windows platforms 133
 - format for maps and map rules 118
 - validating an XML document 51



Printed in USA