

This unit will address the use of commands, functions, variables, and advanced mapping features. A mapping exercise will follow.

Unit Objectives



- Create local and global variables
- Use variables
- Code string and numeric literals in mapping commands
- Use pre-defined special variables
- Use "drag and drop" mapping
- Create mapping commands
- Use any of the Data Transformation mapping functions
- Apply mapping tools to satisfy data translation requirements
- Create Validation maps
- Create new and use existing Functional Acknowledgement maps

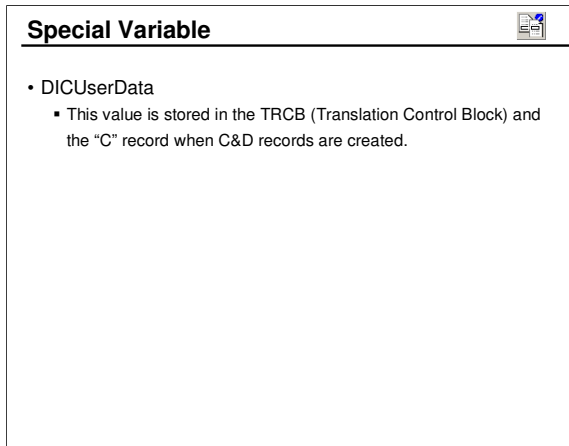
This unit will address each of these objectives.

Variables	
• Variable Types	
▪ Local	
▪ Global	
▪ Special	
• Data Types	
▪ Character	
▪ Integer	
▪ Real	
▪ Binary	
▪ Boolean	
• Scope	
▪ Local – Transaction, Loop	
▪ Global – Group, Interchange	

Variable names can begin with “DI”, but it is not recommended.

Local variables are variables that are not known outside the scope of the translation of a given document.

A local variable “exists” only for the duration of a Loop or Document. A Loop variable is created and initialized at the beginning of each iteration of a loop. Global variables exist for a Group of interchanges, an Interchange of transactions/messages, or for an entire session. A session is defined as beginning when the Data Interchange Services Translator is started and ending when the translator terminates. For example, a global variable might be used to accumulate totals for the



Special Variables are a group of predefined variables used by Data Interchange Services. They function much like Local Variables or Global Variables, except they each have a special purpose. A user can view properties of a Special Variable, but no changes can be made. Special Variable names will always start with "DI". It is recommended that you do not start Local Variables or Global Variable names with "DI". Currently DICUserData is the only Special Variable available.

Literals

- Character Strings
 - Enclosed in quotes ("Virtual Reality") or apostrophes ('WebSphere') or "don't" or 'He said, "yes!"'
- Numeric Values
 - 3.141592737
 - -22

Literal strings are always enclosed in single or double quotes. Remember that keywords like *True* or *False* are not enclosed in quotes.

Keywords



- True

- False


Used to test and set Boolean Values

- This

References the current element

Mapping commands, logical operators, comparison operators, and arithmetic operators are also considered keywords.

Note that these are keywords and not placed in quotes.

Commands 

- Data Mapping
 - Assignment
 - MapTo
 - MapFrom
 - SetProperty
 - Create
- Loop Qualification
 - Quality/Default
 - HLQualify/HLDefault
 - CloseOccurence
 - ForEach
- Logic/Conditional Processing
 - If/Elseif/EndIf
- Error Handling
 - Error
 - FAError
- Hierarchical Loops
 - HLLLevel
 - HLAutoMapped
- Invoking other maps
 - MapCall
 - MapChain
 - MapSwitch
- Schema Support
 - SetNamespace
 - SetSchemaLocation
 - SetNoNSSchemaLocation

Generally commands are selected from a list of commands available at a particular point in the mapping. Commands only appear in the command pane of the mapping window.

Assignment Command



- *targetpath = expression*
 - Function: Establishes a value for a target path
- where:
- targetpath – Path being mapped in the target document.
 - expression – Source path or expression to be evaluated with the result mapped to the target element.
- Note: Ensure that the target path is always on the left side of the assignment statement and that the source path is on the right side.

MapTo Command

- MapTo (*targetpath, expression*)
- Function: Defines an expression and the target path to which it is to be mapped

where:

- targetpath – Path being mapped in the target document.
- expression (optional) – Expression to be evaluated with the result mapped to the target element.

- Note: The MapTo command is used only in source based maps and is produced automatically when elements are mapped via drag and drop.

When you drag a compound element onto a repeating element in the source document definition and there are no existing qualifications for the element, a MapTo command will be inserted under the corresponding repeating element in the Mapping Command window pane. For target based maps, if there are no existing qualifications for the element, a ForEach command will be inserted under the corresponding repeating source element in the Mapping Command window pane.

MapFrom Command


- MapFrom (*targetpath*) - or -
MapFrom (*targetpath*, *expression*)
- Function: Defines an expression to be mapped to the current target element
where:
 - *expression* (optional) – Expression to be evaluated with the result mapped to the target element.
- Note: The MapFrom command is used only in target-based maps and is produced automatically when elements are mapped via drag and drop.

You can use the MapFrom command in the following ways:

- To move data from a repeating simple element in the target document definition to a corresponding repeating simple element in the source document definition. When you use the MapFrom command on a repeating target node, multi-occurrence mapping is required, and the MapFrom command must be included within a ForEach command.

- To move data from a non-repeating simple element or variable in the target document definition to a corresponding simple element in the source document definition.


You cannot use the MapFrom command with compound elements.

SetProperty Command	
<ul style="list-style-type: none">• SetProperty (<i>PropertyName</i>, <i>PropertyValue</i>)• Function: Sets a Property such as EDI envelope element or XML prolog to a value <p>where:</p> <ul style="list-style-type: none">▪ <i>PropertyName</i> – Name of the property being set.▪ <i>PropertyValue</i> – The value to which the named property is to be set.	

The *PropertyName* must be passed as a character string. It may be in a variable or specified as a literal.

For example: SetProperty”ISA05”,”01”).

The SetProperty command is used to set a special processing property of the target message.

Create Command 

- Create (*targetpath*)
- Function: Forces the creation of the specified compound element in the target document

where:

- *targetpath* – Compound element defining the *targetpath* to be created.

Typically this command is not needed as elements are automatically created when data is inserted into them. However, there are cases where an element must be created whether or not data is inserted into it. The *Created* function may be used to determine if the path was created before executing this command.

Qualify/Default Command

- Qualify (*boolean-expression*)
- HLQualify (*boolean-expression*)
- Function: Causes a series of commands to be executed based on a Boolean expression being true
where:
 - *boolean-expression* – A true value will cause the commands within the qualification to be executed.
- The Default and HLDefault commands specify commands to be executed if none of the Qualify expressions evaluate to True.

The Qualify command is used on repeating compound or repeating simple elements in the source document definition. It is used to indicate that a specific iteration or iterations of the elements are to be handled differently than other iterations of the element. For instance, you might want to say that the first iteration of a loop is handled differently than all other iterations of the loop.

The Default command is used to specify the commands that should be executed if none of the Qualify expressions evaluate to True.

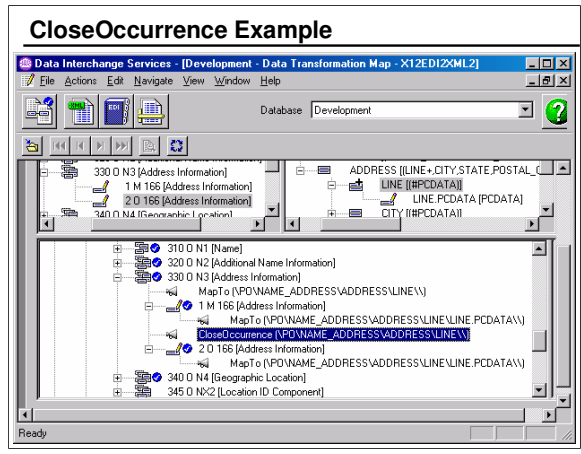
CloseOccurrence Command

- CloseOccurrence (*target-path*)
- Function: Terminates an occurrence mapping

where:

- *target-path* – The repeating target element that is to be “closed” to new elements in the current occurrence. Additional mappings to this element or its children result in a new occurrence of the *targetpath* element.

Use the CloseOccurrence command to close the current occurrence of a repeating element and force the creation of another instance of the element. This command can be used for both source-based and target-based maps. It is used when separate (non-repeating) instances of source data are to be mapped to repeating target data.



In this example there are two separate elements containing address line information in the EDI data. However, in the XML data, the address information is in a repeating element. If each of the two EDI address elements were dragged to the repeating XML target, the second address line would overlay the first. To prevent this, a CloseOccurrence command is issued between the two mappings.

ForEach Command

- ForEach (*source-path*)
- Function: Established a correspondence between the recurring node and the source path specified

where:

- *source-path* – Identifies the source element corresponding to a repeating target element.

- When source nodes are mapped in target-based maps.

ForEach commands are generated by the mapping editor.

Use the ForEach command to execute mapping commands for each occurrence of the specified source node. Each occurrence of the source results in a new occurrence of the current target element.

You can include multiple ForEach command blocks within a single target element. The target element can be either a repeating node or a non-repeating node, and can be either a simple or a compound node.

If the target node is not repeating, you might need to use the Qualify command or conditional logic (If/Then/Else) so that only one occurrence of the source value is written to the target. If the target is not repeating and multiple values are written to it, later values will overwrite the earlier values.

Conditional Commands
<pre>If (<i>condition</i>) <i>commands</i> Elseif (<i>condition</i>) <i>commands</i> Elseif (<i>condition</i>) <i>commands</i> Else <i>commands</i> Endif</pre>

The *If* command marks the beginning of the *If* condition block. *EndIf* is used to mark the ending of the *If* condition block.

When the *If* command is encountered by the translator, the condition will be evaluated. When the *If* condition evaluates to *True*, the mapping commands immediately following the *If* command will be executed. When the *If* condition evaluates to *False*, the translator will look for an *ElseIf* statement within the *If* condition block. If an *ElseIf* statement is found, its associated condition will be evaluated. When condition evaluates to *True*, the mapping commands immediately following the *ElseIf* statement will be executed. When the *If* condition evaluates to *False*, the translator will look for the next *ElseIf* statement within the *If* condition block

Error Command



- Error (*level, code, message*)
 - Function: Issues an error message and sets the return code based on a condition occurring during translation
- where:
- level – severity 0, 1, 2, or 3 (extended error code and JCL condition code)
 - code – unique error code 5000 to 5999
 - message – text message – message to be written as a TR0026 message.

Use the Error command to issue an error condition. This command allows you to establish your own errors for a translation. Typically, the error is issued from within an If conditional block.

FAError Command

- FAError (*level, code, facode, message, error type, segment ID, element position, subelement position*)
 - Function: Used only in Validation Maps to define an EDI Functional Acknowledgement error based on a mapping condition
- where:
- level – severity 0, 1, or 2 (none, element, segment)
 - code – unique error code 5000 to 5999
 - facode – the EDI standard functional acknowledgement code
 - message – text message
 - error type (optional) – I, G, T, S, or E, indication Interchange, Group, Transaction, Segment or Element error
 - segID (optional) – ID of the segment in error
 - element position (optional) – position of the element in error
 - subelement position (optional) – position of the subelement in error

Use the FAError command to set a functional acknowledgement code. You can only use this command in validation maps.

Use this command at any point at which you want to identify an error condition. This command allows you to establish your own error codes for translation. Typically, the error is issued from within an If conditional block.

HLLevel and AutoMapped Command

- HLLevel (*level*)
- Function: Defines an HL loop within an EDI ANSI X12 map
where:
 - level – The hierarchical level code defined for this mapping in the hierarchy.
- HLAutoMapped
 - Produced when an HLLevel command is created within an HL qualification

Use the HLLevel command to create an HL loop within an EDI source or target based map.

If you are working with a source-based map:


-Drag a structure or record from the target window and drop it on to the HL loop in the source window. A MapTo command is created under the loop.

- To create the underlying HL nesting levels or children for this HL level, right-click the HL loop and select AddChild, or select AddPeer to create a sibling.

If you are working with a target-based map:

- Right-click the HL loop again and select the ForEach command

- Drag and drop the source path to the command

AddChild and AddPeer Menu Actions 

- AddChild and AddPeer will appear as choices when you add an HL qualification to a hierarchical loop.
- AddChild
 - Inserts an HL loop as the child of the current loop within an HL qualification.
- AddPeer
 - Inserts an HL loop as a peer to the current loop within an HL qualification.

- To create a child node, select the “AddChild” command.

After selection of AddChild, select an HL level code from the drop-down list. The list contains valid codes from the code list for HL03 (element 735). The HL loop is inserted as a child of the current HL loop with an HLLevel command. The HL segment is automatically selected and all elements mapped with an HLAutoMapped command. This command can not be removed or modified. Additional mapping for these elements can be accomplished using normal mapping methods.

- To create a sibling node, select the “AddPeer” command.

After selection of AddPeer, select an HL level code

MapCall Command



- MapCall (*sourcepath, mapname, targetpath*)
 - Function: Causes the document to be translated by another map with control returned to the current map following translation by the second map
- where:
- *sourcepath* – The source element to be used as the root of the source tree for the imbedded map.
 - *mapname* – The name of the second map by which the current document is to be processed.
 - *targetpath* – The target element identifying the destination of the output of the imbedded map.

Use the MapCall command to indicate that a new map must be used to process the data within the current source element (for source-based maps) or the specified source element (for target-based maps).

When this command is encountered, a new copy of the translator is loaded. It receives the data from the current element to use as its input document. The element can be a simple element (for example, the BIN02 element in a BIN segment) or it can be a compound element (a subtree or subset). The copied translator shares global variables with the parent translator.

When the copied translator has completed translation, it terminates, and control is returned to the parent translator. The parent translator checks if

MapChain Command

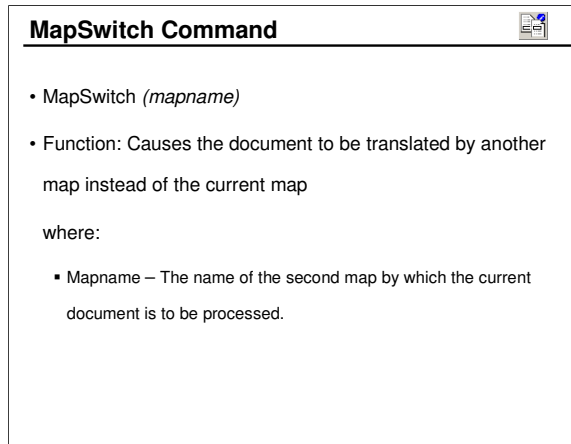
- MapChain (*mapname*)
- Function: Causes the document to be translated by another map following successful completion of the current translation

where:

- *mapname* – The name of the second map by which the current document is to be processed.

Use the MapChain command to indicate that the document needs to be translated by another map after the current translation has completed. The subsequent translation only occurs if the current translation is successful.

Each MapChain command encountered during translation of a map indicates that the document will be translated an additional time using the newly specified map. This process stops if an error is encountered in any one translation. The output from a translation is independent from the other translations.



Use the MapSwitch command to indicate that the document needs to be translated by another map instead of the current map. Any translation performed by the current map is terminated and the document is translated by the map specified in the MapSwitch command.

Use this command when data in the document must be inspected before it can be determined which map to use. It allows you to switch the map dynamically based on the data that is contained in the document. You can create a map that will initially examine the data in the document. Only the compound and simple elements necessary to make a mapping decision are mapped. The map that should be used to translate the document is determined based on the mapped elements. Then use conditional mapping commands

SetNamespace Command

- SetNamespace (*URI*)
- Function: Creates an attribute in the target XML document that associates a prefix to a namespace.
- If this command is included, an attribute of the following form will be created on the root element of the XML output:
 - xmlns:prefix="URI"

Namespace objects are generally created automatically when XML schemas are imported into WebSphere Data Interchange Client. WebSphere Data Interchange Client scans the schema looking for “xmlns” attributes while a schema is being imported. The “xmlns” attribute identifies a namespace. When the attribute is encountered, WebSphere Data Interchange determines if a corresponding Namespace object exists within the same XML Dictionary. If one does not exist, a corresponding Namespace object is created within the XML Dictionary. The prefix and other information associated with Namespace objects may be changed later.

A Universal Resource Identifier (URI) is a member of this universal set of names in registered name.

SetSchemaLocation Command

- SetSchemaLocation (*URI*)
- Function: Creates an attribute in the target XML document that provides information as to where the schema is located. This command is used to generate an attribute that associates the schema with a namespace.
- If this command is included, an attribute of the following form will be created on the root element of the XML output:
 - xsi:schemaLocation="URI location"
- The Namespace object representing the specified URI will be obtained. The location value is filled in based on the information in the Namespace object.

The “xsi” prefix will be changed if you have a Namespace object defined for URI “http://www.w3.org/2001/XMLSchema-instance” with a different prefix. If you do not have a Namespace object defined for this URI, the default prefix “xsi” will be used. An appropriate “xmlns:prefix=http://www.w3.org/2001/XMLSchema-instance” attribute will automatically be generated when this command is used.

This command may be specified multiple times in the map. If there is more than one occurrence of this command, then a namespace/location pair will be created for each occurrence of the command.

SetNoNSSchemaLocation Command

- SetNoNSSchemaLocation (*location*)
- Function: Creates an attribute in the target XML document that provides hints as to where the schema is located. This command is used to generate an attribute that does not associate the schema with a namespace.
- Location – A character expression that indicates the schema location for the target XML document. This can be a string constant, a path identifying a simple element in the source document definition, a variable, or any expression that evaluates to a character string.

The “xsi” prefix will be changed if you have a Namespace object defined for URI “http://www.w3.org/2001/XMLSchema-instance” with a different prefix. If you do not have a Namespace object defined for this URI, the default prefix “xsi” will be used. An appropriate “xmlns:prefix=http://www.w3.org/2001/XMLSchema-instance” attribute will automatically be generated when this command is used.

If there is more than one occurrence of this command, only the last value is used.

Properties



- Properties represent values that may be retrieved or set during translation.
- Source document properties are retrieved using the GetProperty function.
- Target document properties are set using the SetProperty command.
- Property values include envelope elements for EDI data or the XML declaration for XML data:
 - DIProlog – XML Prolog
 - Positional envelope element designations ISAnn, GSnn, STnn, UNBnn, and generic properties such as IchgCtlNum, IchgSndrld, GrpCtlNum, and TrxCode to access envelope elements.

Properties can be accessed using the SetProperty command and GetProperty function.

Source Document Properties



- **Alphanum** - Inbound alphanumeric data Code List
- **BegEnv** - Set to "Y" when the document is the first document in an interchange envelope. Otherwise set to "N".
- **BegGrp** - Set to "Y" when the document is the first document in a group envelope. Otherwise set to "N".
- **Charset** - Inbound character data Code List
- **EndEnv** - Set to "Y" when the document is the last document in an interchange envelope. Otherwise set to "N".
- **EndGrp** - Set to "Y" when the document is the last document in a group envelope. Otherwise set to "N".
- **GrpLvIFA** - Group level functional ack only flag
- **Process** - The process value from the trading partner profile
- **ServSegVal** - The service segment validation level
- **Thandle** - The transaction handle (yyyymmddhhmmssnnnnn)
- **ValLevel** - Inbound validation level
- **ValErrLevel** - Inbound acceptable error level
- **ValMap** - Inbound validation map name

Target Document Properties
<ul style="list-style-type: none"> • ACField – Specifies the application control field • AckReq - Overrides the acknowledgement expected setting • Alphanum - Overrides the outbound alphanumeric data Code List • Charset - Overrides the outbound character data Code List • CtlNumFlag - Overrides the control numbers by transaction id flag • DIDocId - Value to appear in the PRTFILE message when the output is written to a file or queue • DIProlog - Used to override the default XML prolog in the target document • EdiDecNot - Overrides the EDI decimal notation character from the trading partner profile • EdiDeDlm - Overrides the EDI data element delimiter from the trading partner profile • EdiDeSep - Overrides the EDI repetition separator from the trading partner profile

The EDI envelope standard generic properties allow you to get or set information in the EDI envelope standard segments. They have generic names that can apply to any of the EDI standard types. For example, IchgSndrId is used for the interchange sender ID, regardless of the EDI standard type.

These properties are available when the source document is received within an EDI envelope standard. The properties can be obtained using the GetProperty function. If you request a property that is not present, an empty string is returned.

These values can also be set for the target EDI document using the SetProperty command. If set, they will override the values specified in the envelope profile.

Target Document Properties (cont.)



- EdiRlsChar - Overrides the EDI release character from the trading partner profile
- EdiSeDlm - Overrides the EDI subelement delimiter from the trading partner profile
- EdiSegDlm - Overrides the EDI segment delimiter from the trading partner profile
- EdiSegSep - Overrides the EDI segment id separator from the trading partner profile
- EnvProfName - Overrides the envelope profile name
- EnvType - Overrides the envelope type
- NetProfId - Overrides the network profile id from the trading partner profile
- SegOutput - Overrides the segmented output flag from the trading partner profile
- ValLevel - Overrides the outbound alphanumeric data Code List
- ValErrLevel - Overrides the outbound acceptable error level
- ValMap - Overrides the outbound alphanumeric data Code List

Functions		
• Char	• IsEmpty	• StrCompI
• Concat	• Left	• StrCompN
• Created	• Length	• StrCompNI
• Date	• Lower	• SubString
• DateCnv	• Number	• Time
• Exit	• NumFormat	• Translate
• Find	• Occurrence	• TrimLeft
• Found	• Overlay	• TrimRight
• GetProperty	• Right	• Truncate
• HexEncode	• Round	• Upper
• HexDecode	• StrComp	• Validate

All functions take zero or more arguments as input and return a value. The data type of the return value, as well as the number and data types of the arguments varies from one function to the next. Appropriate type conversions are done implicitly if needed (and possible), just as they are done for the expressions. Some functions have optional parameters. If the optional parameters are omitted from the function call, a default value is used for that argument.

Using Functions



- A Function returns a value based on arguments provided an input
- Arguments are specified in parenthesis following the function name:
Char (value)
- Function names are not case sensitive
- Function arguments may be expressions
- The Function reference is replaced with the result returned by the function.

Most functions can take an expression as an argument, as long as the result of the expression is (or can be converted to) the correct data type.

The only time you cannot use an expression as an argument is when the argument is identified as a source or target path. For example, you cannot pass an expression to the Created function.

The input arguments for a function are never modified by the function.

The function names are not case sensitive.

Character Processing Functions



- Char (argument) – Converts a numeric value or expression to a string.
- Concat (argument1, argument2) – Concatenates two character string.
- Substring (string, position, length) – Returns a portion (substring) of the string passed, beginning at "position" for a length of "length."
- Find (string, search string, starting position) – Searches a string for a search string, returning the position of the search string in the string searched. Returns 0 if the search string is not found.
- Length (argument) – Returns the length of a character string.
- Left (string, length) – Returns the leftmost *n* characters from a string.
- Right (string, length) – Returns the rightmost *n* characters from a string.
- Overlay (string1, string2, position, length) – Overlays string1 (or a portion of string1) with string2. Position indicates the position in string2 at which string1 is to be overlaid.
- Lower (argument) – Converts a string to lower case characters.
- Upper (argument) – Converts a string to upper case characters.

Char - This function is not normally needed, since the conversion is generally done implicitly.

However, it can be used to force the conversion to a character value, or to clarify when the conversion is done.

Substring - Only available characters from the string value will be copied. Therefore, the substring will contain length characters unless string has less than position plus length minus one characters.

Left - Only available characters from "string" will be copied. Therefore, the returned string will contain "length" characters unless "string" has fewer than "length" characters.

Right - Only available characters from "string" will be copied. Therefore, the returned string will contain

String Comparison Functions

- Format: StrComp (string1, string2)
- Function: Compares two character strings, returning a 0 if the strings are equal. If string1 is less than string2, a -1 is returned. If string2 is less than string1, a 1 is returned.
- Example: StrComp (alpha, omega)
 - If variable alpha has a value of 'ABC' and omega has a value of 'XYZ,' a -1 will be returned.
- Function
 - StrCompI - Compares two string ignoring the case
 - StrCompN - Compares the first N characters of two strings
 - StrCompNI - Compares the first N characters of two strings ignoring the case

StrComp - The syntax “If (X = ‘123’)” may NOT be used for string comparison. The StrComp function must always be used.

String cannot be compared using an expression such as(string1 = string2)

String Trimming Functions



- Format: TrimLeft (string, trim-string)
- Function: Removes characters (typically blanks) from the left of a string. Characters in the (optional) trim-string are removed from the string. If trim-string is not specified, leading whitespace characters (blanks, tabs, line-feeds, carriage-returns) are removed.
- Examples:
 - TrimLeft (name) - If variable *name* contains " Julie," the string "Julie" is returned.
 - TrimLeft (address, "0123456789 ") - If variable *address* contains "9010 Lake Sunset Drive," the string "Lake Sunset Drive" is returned.
 - TrimRight - Provides the same function for trailing blanks or specified characters.

TrimLeft - All characters contained in trim-string will be removed from the beginning of the value string. The operation ends when the first character not contained in trim-string is encountered. If trim-string is omitted, whitespace is removed from the beginning of the value string. Whitespace includes blanks, carriage returns, line feeds and tab characters.

TrimRight - All characters contained in trim-string will be removed from the end of the value string. The operation ends when the first character not contained in trim-string is encountered. If trim-string is omitted, whitespace is removed from the end of the value

string. Whitespace includes blanks, carriage returns, line feeds and tab characters.

Date and Time Functions

- Date() – Returns the system date in the format *yyyymmdd*.
- DateCnv (source, source mask, target mask) – Converts a date from one format to another.
 - Example: DateCnv(TODAY, 'ccyyymmdd,' 'tm dd, ccyy')*
- Time() – Returns the system time in the format *hhmmss*.

*A complete list of date code may be found in Appendix C of the *WebSphere Data Interchange User's Guide*

Use the right click popup values menu to get a list on common values and a complete list if substitution values that can be used to construct any specific format.

Hex Character Processing Functions

- HexEncode (argument) – Converts a binary value or expression to a hex character string.
- HexDecode (argument) – Converts a hex character string representing a hex value to the equivalent binary value.
- Example:
 - HexEncode (number)
 - The function returns a character string representing the hex code for the binary value in variable number.
 - If the variable number has a value of 255, the HexEncode function will return a character string of "FF."

HexEncode - Each byte of the binary value will be encoded as two characters in the resulting string. For example, if the input is a 4 byte binary value: 0x01020A0B, the result would be the 8-character string: "01020A0B".

HexDecode - Each byte of the binary value will be derived from two characters in the encoded string. For example, if the input is the 8-character string: "01020A0B", the result would be a 4 byte binary value: 0x01020A0B.

Mapping Query Functions



- Created (argument)
 - Determines if a path was created in the target data.
 - Returns Boolean *true* or *false*.
- Found (argument)
 - Determines if data exists in the source element.
 - Returns Boolean *true* or *false*.
- IsEmpty (argument) –
 - Returns a Boolean *True* if the string is empty (null - not blank), otherwise returns a *False*.
- Occurrence (sourcePath) –
 - Provides the occurrence number of a repeating simple or composite element.

Created - This function can only be used with paths in the target document definition. If the element is within a repeating compound element (such as a repeating loop or segment), then only the current (or most recent) iteration is searched for the targetPath.

Found - This function can only be used with paths in the source document definition. If the element is within a repeating compound element (such as a repeating loop or segment), then only the current iteration is searched for the sourcePath.

IsEmpty - If the argument contains a value other than an empty string, the mapping command is executed by the translator.

Occurrence - Use the Occurrence function to obtain the current occurrence number of:

Exit Function



- Format: Exit (exitname, parameter1, parameter2, parameter3, parameter4)
 - Function: Converts a value to a string that will be passed to the field exit in the User Exit profile. Up to 4 parameters can be passed to the Exit function
- where:
- exitname - The name of a User Exit profile that is used as a logical name for the user-provided program or exit routine in the translator.
 - parameters - Optional string values that are passed to the field exit.

Exit is used to invoke a field exit and returns a string value. Results that are returned can be used to update a variable or target a simple element in an assignment statement. The name that contains the function information to be executed.

You must define field exits in the User Exits profile. If the exit is not properly defined, an error is issued. Zero to four parameters can be passed to the exit and it is the job of the field exit to determine how many parameters to expect.

GetProperty Function



- Format: GetProperty (argument)
- Function: Returns the value of the Property Name specified (such as an envelope element).
 - Example:
 - GetProperty ("ISA06") - Returns the value from ISA06 in the X12 Interchange Control Header.

The GetProperty function is used to get a property of the source message. This can be used to retrieve information such as EDI envelope header elements.

Numeric Processing Functions

- Number (argument) - Converts a character string to a real numeric value.
- NumFormat (value, num-decimals, flag) - Converts a numeric value to a character string. The flag is optional. "R" indicates round (default). "T" indicates truncate.
- Round (real number, number of decimal positions) - Rounds a numeric value to a specified number of decimal positions.
- Truncate (real number, number of decimal positions) - Truncates a numeric value to a specified number of decimal positions.

Number - This function is not normally needed, since the conversion is generally done implicitly.

However, it can be used to force the conversion to a numeric value, or to clarify when the conversion is done.

NumFormat - The flag indicates whether unused digits are rounded or truncated. Specify "ROUND" to round unused digits or "TRUNCATE" to truncate unused digits. If no flag is specified, the value defaults to "ROUND".

Round - Round returns a numeric (real) value. If the return value will be assigned to a character string and trailing zeros are to be removed, the NumFormat function should be used.

Truncate - This function returns a numeric (real)

Translate Function



• Format: Translate (table-name, direction, value, error, default)

where:

- table-name - Translation Table
- direction - "SOURCE" or "S" to search the Source column of the table, or "TARGET" or "T" to search the Target column of the table.
- value - The search value (character or real)
- error - Boolean true or false. If true, a warning message will be issued if the search value is not found.
- default (optional) - Value to be returned if search value is not found.

If the table specified does not exist, a warning will be issued and it will be treated as a not found condition.

A translation table contains two values for each entry in the table, one called the Source Value and the other called the Target Value. When direction is specified as "SOURCE", an attempt is made to find the search value in the Source Value column in the table. If it is found, the value from the corresponding Target Value column is returned by the function. When direction is specified as "TARGET", an attempt is made to find the search value in the Target Value column in the table. If it is found, the value from the corresponding Source Value column is returned by the function.

Validate Function

- Format: Validate (table-name, value, error)

where:

- table-name - Code list
- value - the search value
- error
 - Boolean true or false.
 - If true, a warning message will be issued if the value is not found in the table.
- Function: Performs a Code List lookup, returning a Boolean true if the value is found in the table or a false if the value is not found.
- Example:
 - Validate ("CODELIST", cust_code, False) Code List table "CODELIST" will be searched to locate the value cust_code. If found, a true is returned by the function. If not found, no message will be issued (false) and the value false will be returned.

If the value is found in the Code List, the mapping command will be executed. If the value is not found in the Code List, an error is issued with error code 5001.

Applied Mapping

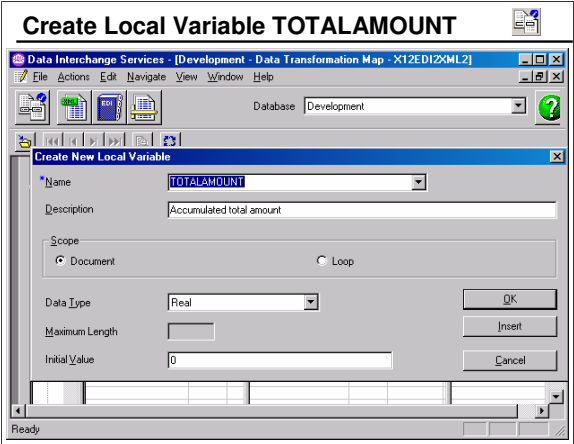


- You've looked at the mapping tools
- Now to apply them
- Next you will learn to:
 - Use variable to perform arithmetic operations, mapping a computed result
 - Map envelope data
 - Create an XML Prolog
 - Convert date formats
 - Use Code Lists
 - Use Translation Tables
 - Manipulate character string data
 - Map hierarchical loops
 - Map conditionally based on inbound qualifying elements

Using Variables



- You have been asked to calculate the total monetary amount for all line items of an inbound purchase order and compare that amount to the amount in the AMT segment.
- To calculate the amount for each line item, it will be necessary to multiply the quantity by the amount.
- If the amount calculated does not match the amount in the AMT segment, you are to issue an error message.



Elements to Be Processed

Data Interchange Services - [Development - Data Transformation Map - XI2ED12XML2]

File Actions Edit Navigate View Window Help

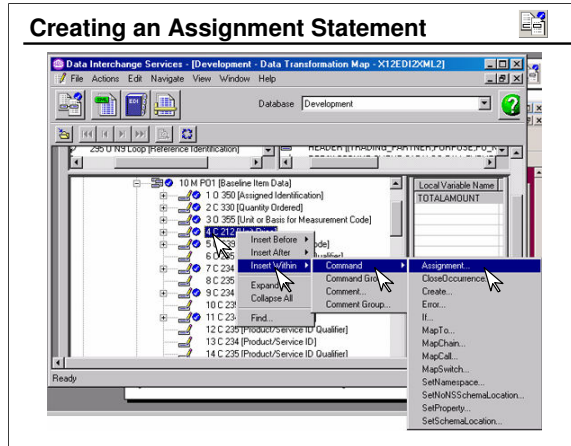
Database: Development

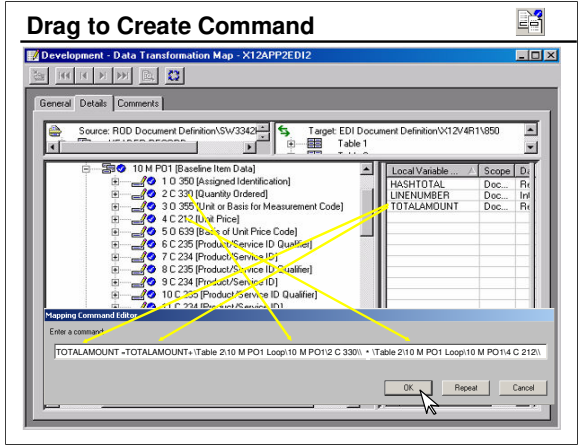
236 U NS Loop (Preference Identification) HEADER (THADING_PARTNERZ_OR_POSC_PO...) Local Variable Name: TOTALAMOUNT

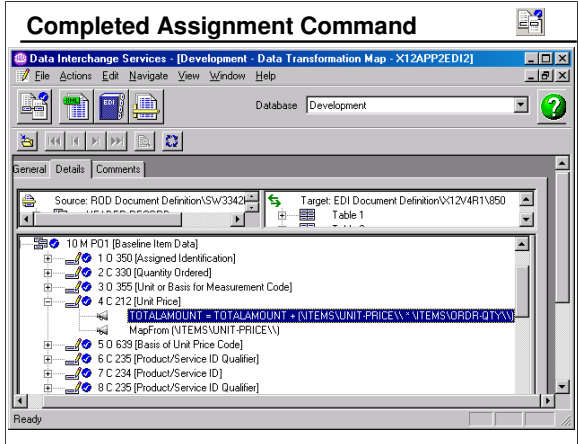
- 10 M P01 [Baseline Item Data]
 - 1 N 350 [Assignment Identification]
 - 2 C 330 [Quantity Ordered]
 - 3 U 639 [Ratio or Unit Price Code]
 - 4 C 212 [Unit Price]
 - 5 U 639 [Ratio or Unit Price Code]
 - 6 C 235 [Product/Service ID Qualifier]
 - 7 C 234 [Product/Service ID]
 - 8 C 235 [Product/Service ID Qualifier]
 - 9 C 234 [Product/Service ID]
 - 10 C 235 [Product/Service ID Qualifier]
 - 11 C 234 [Product/Service ID]
 - 12 C 235 [Product/Service ID Qualifier]
 - 13 C 234 [Product/Service ID]
 - 14 C 235 [Product/Service ID Qualifier]

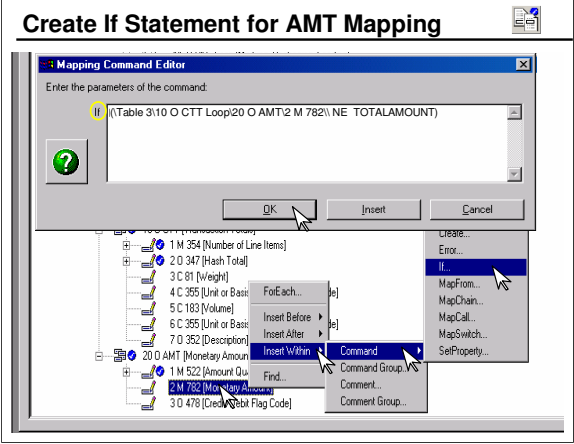
Ready

Creating an Assignment Statement

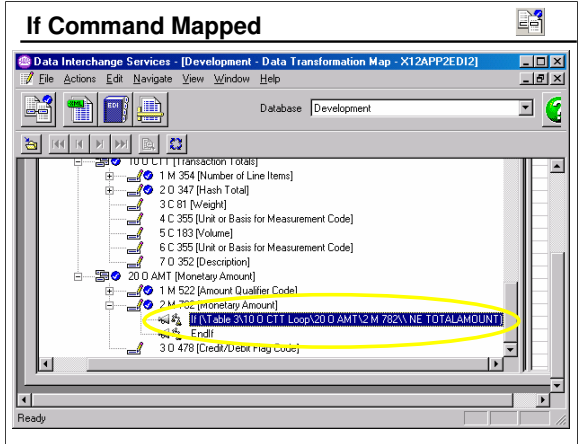


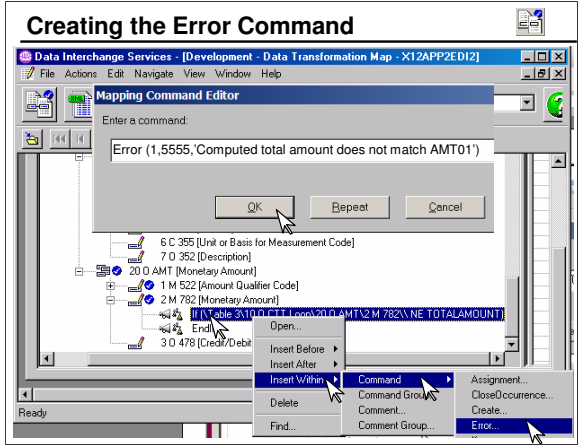






Note that the Mapping Command Editor can be resized if necessary.



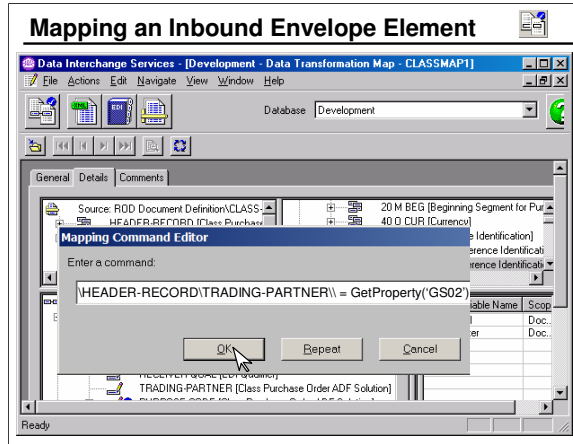


Mapping EDI Envelope Elements

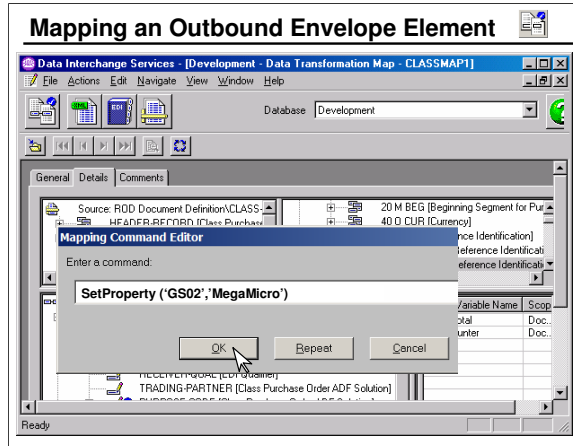


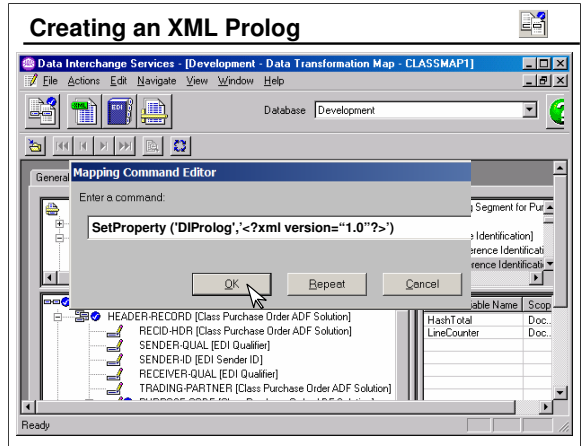
- Frequently there is a requirement to map envelope elements to XML or application data.
- Data Transformation mapping provides the `GetProperty` function to extract envelope data from an inbound interchange.
- Although envelopes are produced automatically, it may be necessary to map data to an envelope.
- Data Transformation mapping provides the `SetProperty` command to create envelope data for an outbound interchange.
- Next, you will look at an example of each type of mapping.

Mapping an Inbound Envelope Element

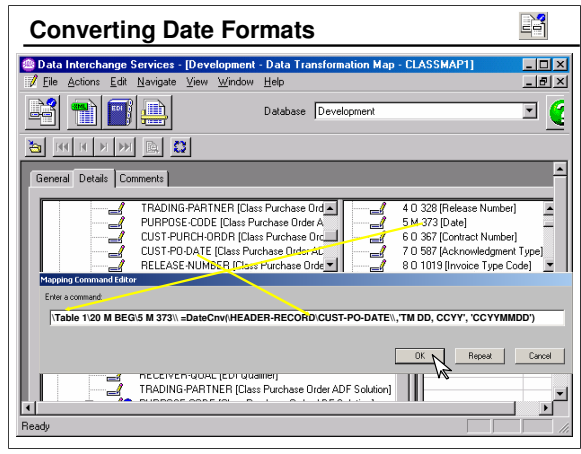


Mapping an Outbound Envelope Element





In this example, Prolog is a variable that will contain the text for the XML Prolog statement.



Remember - TM stands for Textual Month.

Date Masks



CC Century	SS Second of minute
YY Year	WW Week of Year (1 through 52)
MM Month of year	K Day of Week (Monday=1, Tuesday=2, etc.)
DD Day of month	JJJ Julian day of year
D Day of month as a single character, if possible*	Q Quarter (1,2,3,4)
HH Hour of day	E Semester
MM Minute of hour**	ZZZ Time zone
ll Minute of hour	TM Textual month (i.e., January, February, etc.)

*D can be used if it immediately follows WW as in WWD; however, if you want day of week followed by week, you must use KWW, because DWW would be interpreted to be day of month and Week of year.

**MM can be used for minutes when it immediately follows HH as in HHMM; however, if you want minute followed by hour, you must use lHH, because MMHH would be interpreted as month of year and Hour of day.

Using a Code List

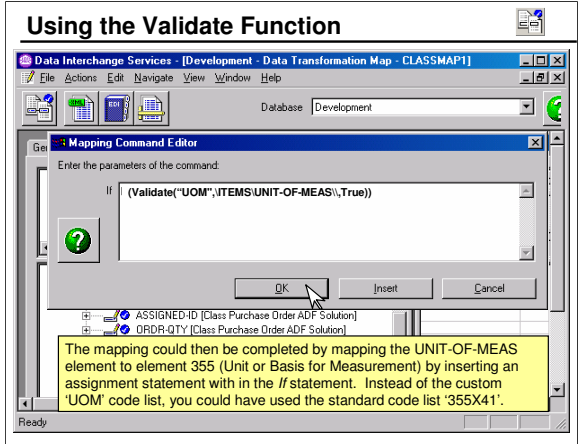
The screenshot shows the 'Data Interchange Services' application window. The title bar reads 'Data Interchange Services - [Development - Data Transformation Map - CLASSMAP1]'. The menu bar includes 'File', 'Actions', 'Edit', 'Navigate', 'View', 'Window', and 'Help'. The 'Database' dropdown is set to 'Development'. The main workspace is divided into 'General', 'Details', and 'Comments' tabs. The 'General' tab is active, displaying a list of data elements and their corresponding code lists:

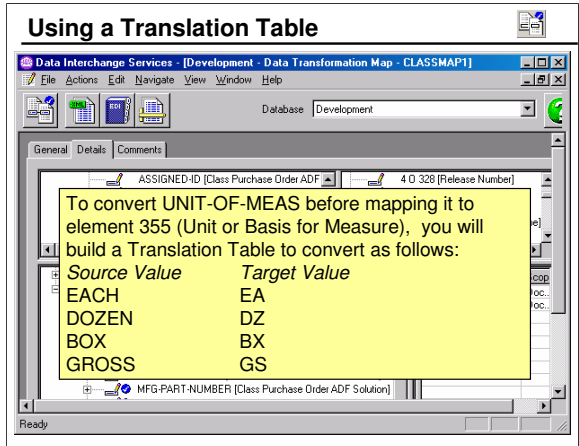
Element Name	Code List
ASSIGNED-ID [Class Purchase Order ADF	4 O 328 [Release Number]
ORDR-QTY [Class Purchase Order ADF Sc	5 M 373 [Date]
UNIT-OF-MEAS [Class Purchase Order AD	6 O 387 [Contract Number]
UNIT-PRICE [Class Purchase Order ADF S	7 O 587 [Acknowledgment Type]
UNIT-PRICE-BASIC [Class Purchase Order	8 O 388 [Contract Number]

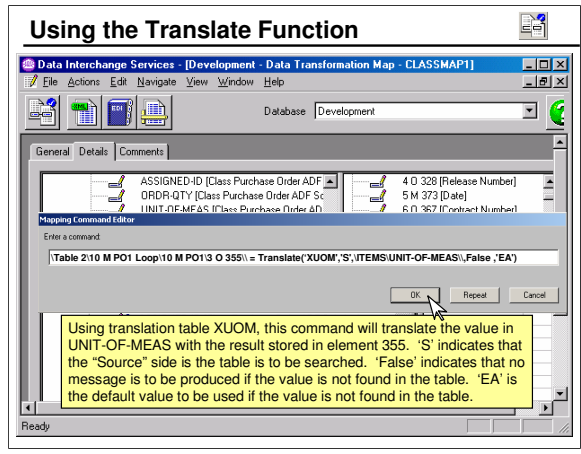
A yellow callout box is overlaid on the 'UNIT-OF-MEAS' row, containing the following text:

To validate UNIT-OF-MEAS before mapping it to element 355 (Unit or Basis for Measure), you will build a custom code list 'UOM' with the following entries: EA, DZ, BX, GS.

The status bar at the bottom left shows 'Ready'.







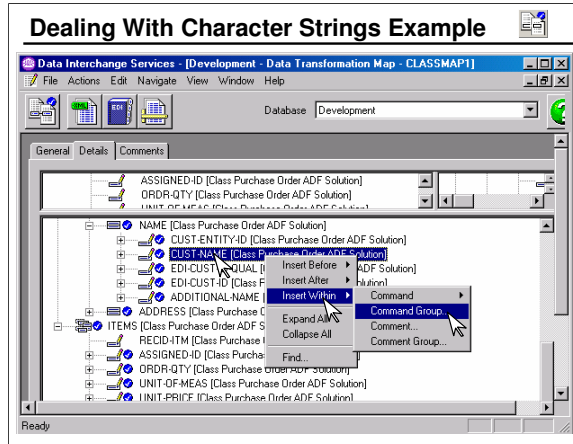
“T” would indicate that the Target side of the table is to be searched.

Dealing with Character Strings



- This example has names in the application data in the form:
 - Firstname Lastname
- You need to convert those names to XML data in the form:
 - Lastname, Firstname
- You will enclose those commands in a "Command Group."
- Creating a Command Group will provide you with an opportunity to describe the Command Group.
- You will then insert the group of related commands under this description.
- You also need to create some Character and Integer local variables to assist with this mapping.

Dealing With Character Strings Example



Character Manipulation in a Command Group

The screenshot shows a software interface for XML transformation. The main window displays a tree view of the transformation process. A yellow circle highlights a specific command group within the 'ShipTo' element. The command group is titled 'Convert name from "First Last" to "Last, First"'. Below this title, several commands are listed, each with a small icon to its left:

- FIRSTLAST = SHIPTO(\\NAME\\)
- BLANKPOS = Find(FIRSTLAST, ',')
- FIRST = Left(FIRSTLAST, BLANKPOS - 1)
- NAMELENGTH = Length(FIRSTLAST)
- LAST = Right(FIRSTLAST, NAMELENGTH - BLANKPOS)
- LASTFIRST = Concat(LAST, ',')
- LASTFIRST = Concat(LASTFIRST, FIRST)

On the right side of the interface, there is a 'Local Variable Name' table with the following entries:

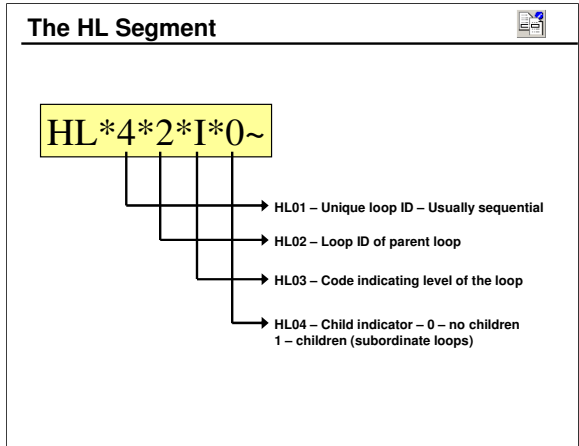
Local Variable Name
BLANKPOS
LAST
NAMELENGTH
LASTLENGTH
FIRSTLAST
FIRST
LASTFIRST

The status bar at the bottom of the window shows 'Ready' and 'NUM'.

Hierarchical Loops

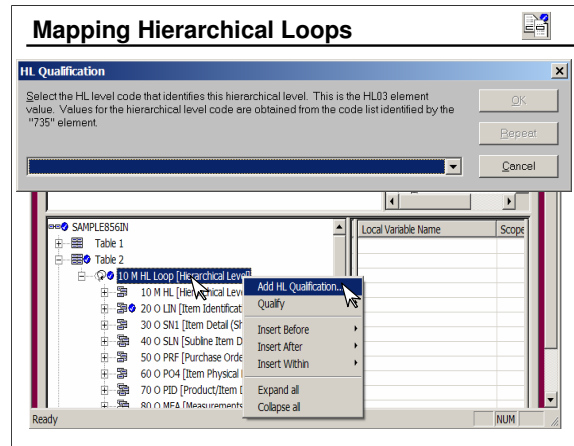


- Hierarchical Loops (HL loops) or “Self-defining loops” allow the user to define the looping structure or hierarchy
- Hierarchical Loops are defined by the HL segment in the ANSI X12 EDI standard
- The HL Segment:
 - Identifies the start of each loop in the hierarchy
 - Assigns an ID number to the loop
 - Identifies the parent (next outer) loop
 - Indicates the nesting level of the loop
 - Indicates whether the loop has inner loops (children)
- HL Qualification is available when the map is based on the EDI standard – target based for inbound, source based for outbound.



A hierarchical loop is similar to an organization chart. Just as an organization chart shows you the various groups of people and their relationships to the whole, a hierarchical loop shows you each group of data and its relationship to the whole.

Hierarchical loops define different levels of data, which can be used in any sequence and skipped when appropriate. This allows you to place the loop anywhere in your data.

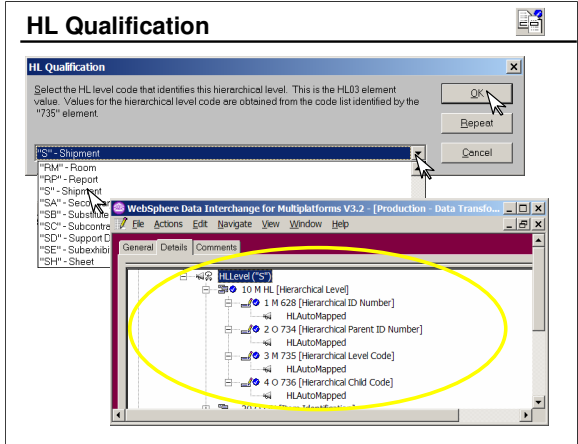


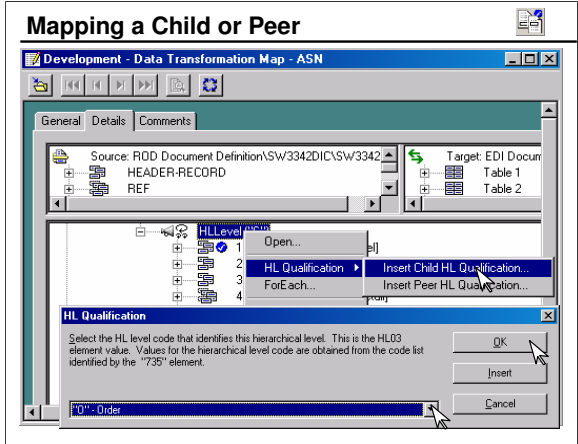
HL01 (ID number) - A unique number that identifies the occurrence of the HL segment. This data element is alphanumeric and has a maximum length of 12 characters. This field usually contains a sequential number that is incremented for each occurrence of the HL segment.

HL02 (Parent ID) - The HL01 value of the HL segment that is the parent of the current HL segment.

HL03 (Level Code) - A code that indicates the level of the HL segment in the current HL loop. For example, the level code could refer to the shipment, order, or item level information in the ANSI X12 Shipping Notice transaction set.

HL04 (Child code) - A code that indicates if the segment has subordinate segments: 1 for subordinate





Mapping Qualified Inbound Elements

- You have seen that, for outbound maps, you can map qualifiers from fields or specify literal values.


UP 593331112377

In this example the 'UP' in the first element indicates that the second element contains a UPC code.

- Now you will look at receiving that qualified data and inspecting the qualifier to determine where the qualified element is to be mapped.
- In the example, if the qualifier is 'UP,' you want to map the qualified element to the UPC code field in the target data.

Qualifier Based Inbound EDI Mapping

The screenshot displays an EDI mapping interface with the following components:

- Source:** EDI Standard Transaction (V12V4R1) 856
- Target:** Data Format (CLASS-ADF_D0)

Source Structure:

- Table 1
- Table 2
 - 10 M HL Loop (Hierarchical Level)
 - 10 M HL (Hierarchical Level)
 - 20 O LIN (Item Master Data)
 - 1 C 350 (Assigned Identification)
 - 2 M 235 (Product/Service ID Qualifier)
 - 3 M 234 (Product/Service ID)
 - 4 C 235 (Product/Service ID Qualifier)
 - 5 C 234 (Product/Service ID)
 - 6 C 235 (Product/Service ID Qualifier)
 - 7 C 234 (Product/Service ID)
 - 8 C 235 (Product/Service ID Qualifier)

Target Structure:

- HEADER-RECORD (Class Purch)
- REF (Class Purchase Order ADF)
- NAMEADDRESS (Class Purchas)
- ITEMS (Class Purchase Order Ai)
 - RECID-ITM (Class Purcha)
 - ASSIGNED-ID (Class Purc)
 - ORDR-QTY (Class Purcha)
 - UNIT-OF-MEAS (Class Pur)
 - UNIT-PRICE (Class Purch)
 - UNIT-PRICE-UNIT (Class
 - MPG-PART-NUMBER (Class
 - BUYERS-ITEM-NUM (Class
 - UPC-CODE (Class Purcha)

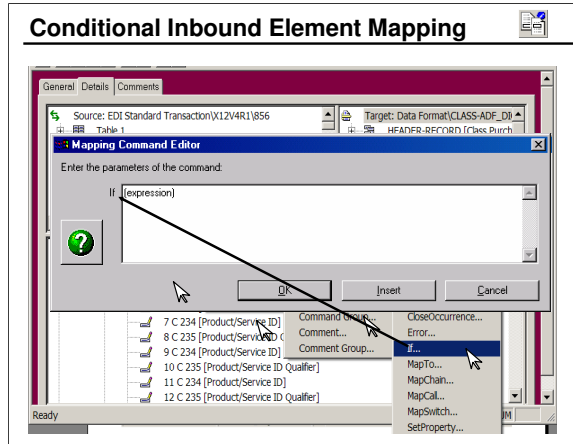
Mapping: The source qualifiers are mapped to the target fields as follows:

- Source 1 C 350 (Assigned Identification) maps to Target ASSIGNED-ID (Class Purc)
- Source 2 M 235 (Product/Service ID Qualifier) maps to Target ORDR-QTY (Class Purcha)
- Source 3 M 234 (Product/Service ID) maps to Target UNIT-OF-MEAS (Class Pur)
- Source 4 C 235 (Product/Service ID Qualifier) maps to Target UNIT-PRICE (Class Purch)
- Source 5 C 234 (Product/Service ID) maps to Target UNIT-PRICE-UNIT (Class
- Source 6 C 235 (Product/Service ID Qualifier) maps to Target MPG-PART-NUMBER (Class
- Source 7 C 234 (Product/Service ID) maps to Target BUYERS-ITEM-NUM (Class
- Source 8 C 235 (Product/Service ID Qualifier) maps to Target UPC-CODE (Class Purcha)

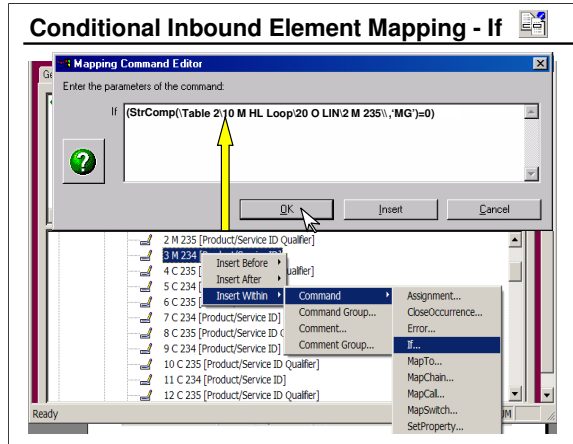
Other Elements:

- Bottom left: SAMPLE-SOURCEBAS
 - Table 1
 - Table 2
 - Table 3
- Bottom status: Ready
- Bottom right: NUM

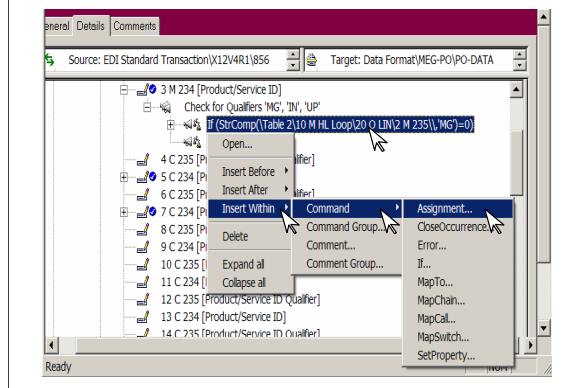
Conditional Inbound Element Mapping



Conditional Inbound Element Mapping - If



Conditional Inbound Element Mapping Command



Conditional Mapping Based on 'MG' Qualifier

The screenshot displays a software interface for mapping EDI data. The main window shows a tree view of the source data structure, including 'Table 1', 'Table 2', and a '10 M HL Loop' containing '10 M HL' and '20 O LIN' with sub-qualifiers '1 O 350', '2 M 235', and '3 M 234'. A 'Mapping Command Editor' dialog box is open in the foreground, with the following text:

Enter a command:
ITEMS\MFG-PART-NUMBER\ = \Table 2\10 M HL Loop\20 O LIN\3 M 234\
OK Repeat Cancel

Yellow arrows indicate the mapping logic: one arrow points from the '3 M 234' qualifier in the source tree to the '3 M 234' part of the command, and another arrow points from the 'MFG-PART-NUMBER' field in the target list to the 'MFG-PART-NUMBER' part of the command.

Conditional Element Mappings

The screenshot displays a software interface for defining conditional mappings. The source is identified as 'EDI Standard Transaction(X12V4R1)856' and the target is 'Data Format(CLASS-ADF_DICTIONARY)CLASS'. The source structure includes 'Table 1', 'Table 2', and a '20 O LIN [Item Identification]' loop. The target structure includes 'HEADER-RECORD', 'REF', 'NAMEADDRESS', and 'ITEMS'. The mapping for '20 O LIN' is detailed below:

```
20 O LIN [Item Identification]
├── 1 O 350 [Assigned Identification]
├── 2 M 234 [Product/Service ID Qualifier]
├── 3 M 234 [Product/Service ID]
│   ├── If (StrComp(Table 2\10 M HL Loop\20 O LIN\2 M 235\, 'MG') = 0)
│   │   └── \ITEMS\MFG-PART-NUMBER\ = (Table 2\10 M HL Loop\20 O LIN\3 M 234\
│   └── ElseIf (StrComp(Table 2\10 M HL Loop\20 O LIN\2 M 235\, 'IN') = 0)
│       └── \ITEMS\BUYERS-ITEM-NUM\ = (Table 2\10 M HL Loop\20 O LIN\3 M 234\
│   └── ElseIf (StrComp(Table 2\10 M HL Loop\20 O LIN\2 M 235\, 'UP') = 0)
│       └── \ITEMS\UPC-CODE\ = (Table 2\10 M HL Loop\20 O LIN\3 M 234\
│   └── EndIf
└── 4 C 234 [Product/Service ID Qualifier]
```

A yellow oval highlights the conditional logic block (lines 2 through 4) in the mapping tree.

Additional Mappings



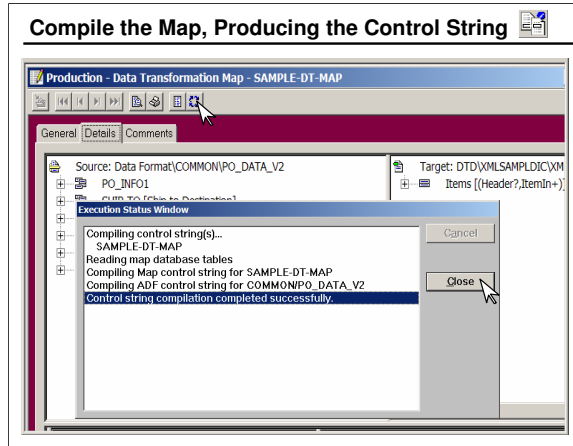
- You have just completed the mapping for the first 235-234 element pair, checking for any of three qualifiers.
- However, there are nine more potential pairs of qualifiers and IDs that may need to be mapped.
- The Client allows you to shift and drag mappings from one element to another element.
- You can drag the mappings from the first element 234 mappings to the next.
- To facilitate this, you can first create a group and copy those mappings as a group.
- Then you can update the path references by opening the mapping editor and dragging the appropriate qualifier and ID elements to the new copies.
- This provides function similar to &SAMEAS with Send/Receive maps.

Completed Conditional Element Mappings

General Details Comments

- 20 O LIN [Item Identification]
 - 1 O 350 [Assigned Identification]
 - 2 M 235 [Product/Service ID Qualifier]
 - 3 M 234 [Product/Service ID]
 - Check for Qualifiers 'MG', 'IN', 'UP'
 - If (StrComp(Table 2\10 M HL Loop\20 O LIN\2 M 235\,'MG')=0)
 - Elseif (StrComp(Table 2\10 M HL Loop\20 O LIN\2 M 235\,'IN')=0)
 - Elseif (StrComp(Table 2\10 M HL Loop\20 O LIN\2 M 235\,'UP')=0)
 - Endif
 - If (StrComp(Table 2\10 M HL Loop\20 O LIN\4 C 235\,'MG')=0)
 - Elseif (StrComp(Table 2\10 M HL Loop\20 O LIN\4 C 235\,'IN')=0)
 - Endif

Compile the Map, Producing the Control String

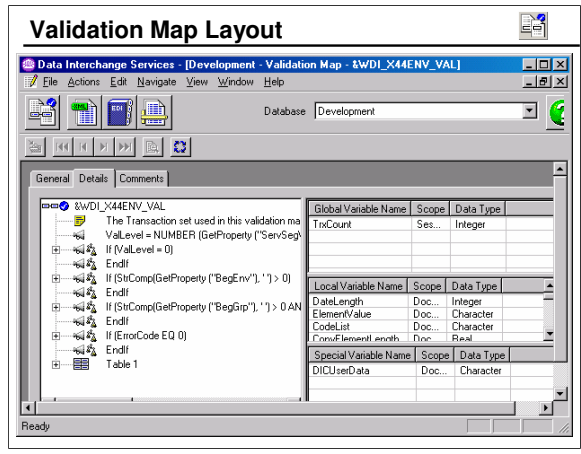


Validation Maps



- Source based
- There is no target document
- FAError may be used
- Many standard mapping techniques may be applied such as the IF and Error commands and the Validate function.
- Code Lists for EDI ID-type elements are included with the distribution materials.
- The following Validation Maps are used to perform service segment validation during data transformation processing:
 - &WDI_E99AENV_VAL-UN/EDIFACT based on E99A service segments and code lists
 - &WDI_UCSENV_VAL-UCS based on UCS 4050 service segments and code lists
 - &WDI_X44ENV_VAL-X12 based on X12 4040 service segments and code lists

Validation maps are only used in conjunction with Data Transformation maps. If the target data is EDI, the Validation map is run after the translation defined by the Data Transformation map is complete. If the source data is EDI the Validation map first validates the inbound EDI data. The Data Transformation then only runs if the validation did not fail.



A validation map is used to handle extended validation requirements, and to call an extended error function to report the information needed to create a functional acknowledgment. A validation map can also copy to local variables, and use most other mapping functions. A Validation map cannot produce an target file.

Functional Acknowledgement Maps



- Source or target based
- Source document is:
\$FUNC_ACK_META
- Source document uses dictionary:
\$FUNC_ACK_METADATA_DICTIONARY
- Target document is defined by an EDI standard
- Target documents provided include:
 - \$DT99724 – For X12 997 Version 2 Release 4 and lower
 - \$DT99735 – For X12 997 Version 3 Release 5 and lower
 - \$DT99737 – For X12 997 Version 2 Release 7 and lower
 - \$DT99933 – For UCS 999 Version 3 Release 3 and lower
 - \$DTCTL – For UN/EDIFACT earlier than Version 94B
 - \$DTCTL21 – For UN/EDIFACT version 94B (Version 2, Release 1) and later.

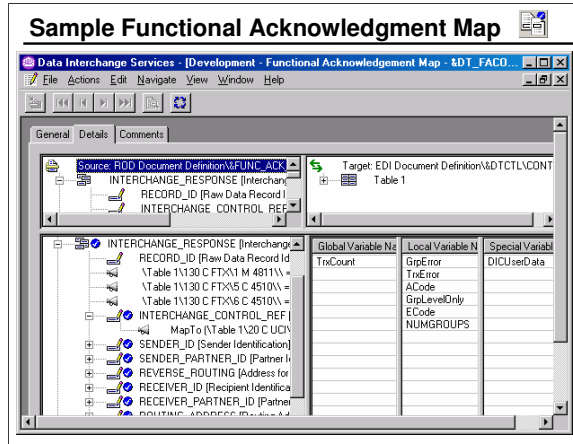
Typically you would select the appropriate map provided for your version and release of the EDI standard. However, if you have special functional acknowledgement requirements, you may wish to write your own Functional Acknowledgement map.

Functional Acknowledgment Maps Provided

	Map Name	Description	Updated Date and Time
1	&DT_FA997V2R4	Functional Acknowledgment 997 - X12V2R4	8/4/2004 10:06:36 AM
2	&DT_FA997V3R5	Functional Acknowledgment 997 - X12V2R4	8/4/2004 10:06:37 AM
3	&DT_FA997V3R7	Functional Acknowledgment 997 - X12V2R4	8/4/2004 10:06:38 AM
4	&DT_FA999V3R3	Functional Acknowledgment 999 - UCSV3R3	8/4/2004 10:06:39 AM
5	&DT_FACONTRL	Functional Acknowledgment CONTRL - Prior to D94B	8/4/2004 10:06:40 AM
6	&DT_FACONTRL94B	Functional Acknowledgment CONTRL - V2R1 (D94B)	8/4/2004 10:06:41 AM
7	&WDI_TA1_ACK	WDI TA1 mapping	10/4/2004 1:33:47 PM

A functional acknowledgement map is a data transformation map that allows you to create a functional acknowledgment to be returned to your trading partner. Both source and target based mapping are available for mapping functional acknowledgments.

Sample Functional Acknowledgment Map



The screenshot displays the 'Data Interchange Services - [Development - Functional Acknowledgment Map - 4DT_FACD...]' window. The 'General' tab is active, showing the source and target document definitions. The source is 'POD Document Definition\FUNC_ACK' and the target is 'EDI Document Definition\DTCTLVCONT'. The map structure includes the following elements:

- Source: POD Document Definition\FUNC_ACK
 - INTERCHANGE_RESPONSE [Interchange]
 - RECORD_ID [Raw Data Record Id]
 - INTERCHANGE_CONTROL_REF [Raw Data Record Id]
- Target: EDI Document Definition\DTCTLVCONT
 - Table 1

The 'MapTo' section shows the following mappings:

- INTERCHANGE_RESPONSE [Interchange]
 - RECORD_ID [Raw Data Record Id]
 - \Table 1\130 C FT\11 M 4811\ =
 - \Table 1\130 C FT\15 C 4510\ =
 - \Table 1\130 C FT\16 C 4510\ =
 - INTERCHANGE_CONTROL_REF [Raw Data Record Id]
 - MapTo \Table 130 C UC\
 - SENDER_ID [Sender Identification]
 - SENDER_PARTNER_ID [Partner Identification]
 - REVERSE_ROUTING [Address for Return]
 - RECEIVER_ID [Recipient Identification]
 - RECEIVER_PARTNER_ID [Partner Identification]
 - ROUTING_ADDRESS [Routing Address]

The 'Global Variable Nz', 'Local Variable Nz', and 'Special Variable' tables are as follows:

Global Variable Nz	Local Variable Nz	Special Variable
TrxCount	GrpError	DICUserData
	TrxError	
	ACode	
	GrpLevelOnly	
	ECode	
	NUMGROUPS	

Unit Summary



- Local and global variables are defined in the lower right pane
- Variables may be used in mapping commands
- String literals are coded in single or double quotes
- Special variables are provided by IBM
- Mapping commands MapFrom and MapTo are produced with drag and drop mapping
- Mapping commands may be entered manually
- Data Transformation mapping functions provide for data interrogation and manipulation
- Mapping tools are applied to satisfy special mapping requirements
- Validation maps are always used in conjunction with Data Transformation maps
- Functional Acknowledgement maps are provided, but custom maps may be created