NaturalONE

Code Generation

Version 9.3.4

October 2026

ADABAS & NATURAL

This document applies to NaturalONE Version 9.3.4 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 2009-2026 Software GmbH, Darmstadt, Germany and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software GmbH product names are either trademarks or registered trademarks of Software GmbH and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software GmbH and/or its subsidiaries is located at https://softwareag.com/licenses.

Use of this software is subject to adherence to Software GmbH's licensing conditions and terms. These terms are part of the product documentation, located at https://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software GmbH Products / Copyright and Trademark Notices of Software GmbH Products". These documents are part of the product documentation, located at https://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software GmbH.

Document ID: NBS-N1CODEGENERATION-934-20261004

Table of Contents

Preface v
1 About this Documentation 1
Document Conventions 2
Online Information and Support 2
Data Protection 3
I Using the Code Generation Component 5
2 Introduction 7
Access the Code Generators 8
3 Create a REQUEST-DOCUMENT Client 11
Introduction to the REQUEST-DOCUMENT Client Code Generator 12
Generate the REQUEST-DOCUMENT Subprogram 12
User Exits for the REQUEST-DOCUMENT Subprogram 18
Define XML Substitution Characters 18
4 Create an Object-Maintenance Process 23
Generate the Object Maint Subprogram 24
User Exits for the Object Maint Subprogram 30
5 Create an Object-Browse Process 33
Introduction to the Object-Browse Code Generator 34
Generate the Object-Browse Subprogram 34
User Exits for the Object-Browse Subprogram 38
6 Create an Object Skeleton Subprogram 41
Generate the Object Skeleton Subprogram 42
User Exits for the Object Skeleton Subprogram 47
7 Regenerate Subprograms and Associated Modules 49
Regenerate a Subprogram and Associated Modules 50
Regenerate Multiple Subprograms 51
Compare Differences 53
8 Set Preferences 55
Set Code Generation Preferences 56
Set Logging Preferences 57
Set Natural Preferences 59
9 Customize the Code Generators 63
Export the Supplied Templates 64
Customize a Supplied Template 67
II Using Natural Construct 69
10 Introduction 71
Supplied Client Generation Wizards 72
Requirements 74
Perform Standard Actions on Natural Construct Resources 76
Use the Dependencies View 81
11 Natural Construct Generation 85
Access the Client Generation Wizards 86
Generate the Modules 87

Common Wizard Specifications and Development Tasks 91
Example of Generating a Program 201
Regenerate Natural Construct-Generated Modules 204
12 Natural Construct Administration 207
Create a New Client Generation Wizard 208
Download Natural Construct Resources to a Local Project 245
Modify an Existing Natural Construct Resource 247
Create and Maintain a Natural Construct Model 247
Create and Maintain a Code Frame 251
Create and Maintain a Natural Construct Model UI 258
13 Set Natural Construct Preferences 271
Set Construct Preferences 272
Set Installation Preferences 274
III 277
14 Defining User Exits 279
Introduction 280
Define a User Exit 280
15 Using the Construct Runtime/Compile Time Modules in Non-Construct Server Environments 293
Add the Construct Runtime Project 294
Update the Construct Runtime Project to the Latest Version 295
Replace the Construct Runtime Project with the Latest Version 296
Exclude Modules from the Update or Replace Process 296
Add Customized Modules to the Construct Runtime Project 297
Build the Construct Runtime Project in a non-Construct Server Environment 298
16 Generating an Ajax Page for Generated Subprograms 301
Generate an Ajax Page for an Object-Browse Subprogram 302
Generate an Ajax Page for an Object-Maint Subprogram 310
Generate an Ajax Main Program from an Adapter File 318
Test the Generated Main Program 322
Regenerate the Main Program 324

Preface

Code Generation describes how to use the code generation components of NaturalONE to generate Natural modules in Eclipse.

This documentation is intended for developers who are familiar with NaturalONE and want to use the code generation components to create Natural subprograms and their corresponding data areas locally.

Code Generation covers the following topics:

Using the Code Generation Component	Describes how to use the Code Generation wizards to generate and regenerate Natural subprograms and their associated modules.
Using Natural Construct	Describes how to use the Natural Construct client generation wizards to generate and regenerate Natural Construct subprograms and their associated modules. It also describes how to maintain Natural Construct (for example, to define a new model, model user interface or code frame). **Note:** You must have Natural Construct installed in a server environment to use this component.
Defining User Exits	Describes the user exits generated by the code generation and Natural Construct wizards, and how to define them.
Using the Construct Runtime/Compile Time Modules in Non-Construct Server Environments	Describes the Construct runtime project for the client, which contains all the required modules to eliminate compile and parsing errors caused by missing Natural Construct resources.
Generating an Ajax Page for Generated Subprograms	Describes how to generate an Ajax page for a subprogram generated by either the Object-Browse-Subp or Object-Maint-Subp wizards.

1 About this Documentation

- Document Conventions 2
- Online Information and Support 2
- Data Protection 3

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

Product Training

You can find helpful product training material on our Learning Portal at **https://learn.softwareag.com**.

Tech Community

You can collaborate with Software GmbH experts on our Tech Community website at **https://tech-community.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software GmbH news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software GmbH resources.

Product Support

Support for Software GmbH products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

I Using the Code Generation Component

Code Generation describes how to use the code generation components of NaturalONE to generate Natural modules in Eclipse.

This documentation is intended for developers who are familiar with NaturalONE and want to use the code generation components to create Natural subprograms and their corresponding data areas locally.

This part describes the Code Generation component supplied with NaturalONE.

The following topics are covered:

Introduction
Create a REQUEST-DOCUMENT Client
Create an Object-Maintenance Process
Create an Object-Browse Process
Create an Object Skeleton Subprogram
Regenerate Subprograms and Associated Modules
Set Preferences
Customize the Code Generators

2 Introduction

- Access the Code Generators 8

This section describes the Code Generation component supplied with NaturalONE and how to access the code generators. The Code Generation component provides wizards that generate the following modules:

Modules	Code Generator	Description
REQUEST-DOCUMENT subprogram and corresponding parameter data areas	REQUEST-DOCUMENT Client	Uses REQUEST DOCUMENT and PARSE XML statements to call an external Web service and interpret the response.
Object-maintenance subprogram and corresponding parameter data areas	Object Maint	Updates all entities within a Natural object.
Object-browse subprogram and corresponding parameter data areas	Object Browse	Provides the browse functionality for a Natural object.
Object skeleton subprogram	Object Skeleton	Provides a starting point to create an object subprogram.

The generated subprograms include a full range of user exits. For information about adding custom code within user exits, see *Defining User Exits*.

Notes:

1. To install the code generators for NaturalONE, **Designer > NaturalONE > Service Development** must be selected in the installation tree for the installer. **NaturalONE > Service Development** is selected by default when you select **Designer** in the installation tree.
2. Although the modules are not generated by Natural Construct, the source code lines in the editor are protected.

Access the Code Generators

Note: The code generators must be initiated from an existing NaturalONE project in the NaturalONE perspective.

To access the code generators

1 Open the context menu in the **Project Explorer** view for the NaturalONE project into which you want to generate the modules.

Or:

Open the context menu in the **Project Explorer** view for the library into which you want to generate the modules.

2 Select **Code Generation**.

The code generators are displayed.

For information on using the client generation wizards for Natural Construct to generate modules locally, see ***Using Natural Construct***.

3 Create a REQUEST-DOCUMENT Client

- Introduction to the REQUEST-DOCUMENT Client Code Generator 12
- Generate the REQUEST-DOCUMENT Subprogram 12
- User Exits for the REQUEST-DOCUMENT Subprogram 18
- Define XML Substitution Characters 18

This section describes how to create a REQUEST-DOCUMENT client.

Introduction to the REQUEST-DOCUMENT Client Code Generator

The REQUEST-DOCUMENT Client code generator allows Natural to access Web services by generating a REQUEST-DOCUMENT subprogram based on a Web service WSDL and XSD. In addition, the generator creates a subprogram for each operation (method) in the WSDL and parameter data areas (PDAs) containing parameters that represent the request and response portions of the Web service operation.

The generated REQUEST-DOCUMENT subprogram uses Natural REQUEST DOCUMENT and PARSE XML statements to call the Web service and interpret the response. The subprogram then maps the input parameters to an XML file, which is sent to the Web service via a REQUEST DOCUMENT statement. The response is verified and parsed in the REQUEST-DOCUMENT subprogram and the data is placed into the appropriate output parameters of the PDA. In addition, the generated error PDA informs users of any errors.

A wizard also performs a pre-analysis of the WSDL. If an associated operation requires more than three dimensions, the operation will be disabled on the selection panel because Natural can only handle up to three dimensions. The pre-analysis wizard also checks for cyclic types (a type that is defined in the WSDL and then referenced by another type in the same WSDL). If a cyclic type is found, all operations that reference it will also be disabled.

You can use a REQUEST-DOCUMENT subprogram to perform various functions, such as retrieve the current exchange rate for orders, verify that a postal code and address match, or retrieve inventory information from another application (within or outside the company). The generated subprogram supports Unicode characters, binary arrays and complex structures (arrays of ANY, detailed arrays, etc.).

Note: To use this feature, the Natural nucleus/profile must be set up to correctly handle XML. For information, see *Activate REQUEST DOCUMENT Statement* and *Activate PARSE XML Statement* in the Natural documentation.

Generate the REQUEST-DOCUMENT Subprogram

To generate a REQUEST-DOCUMENT subprogram and data areas

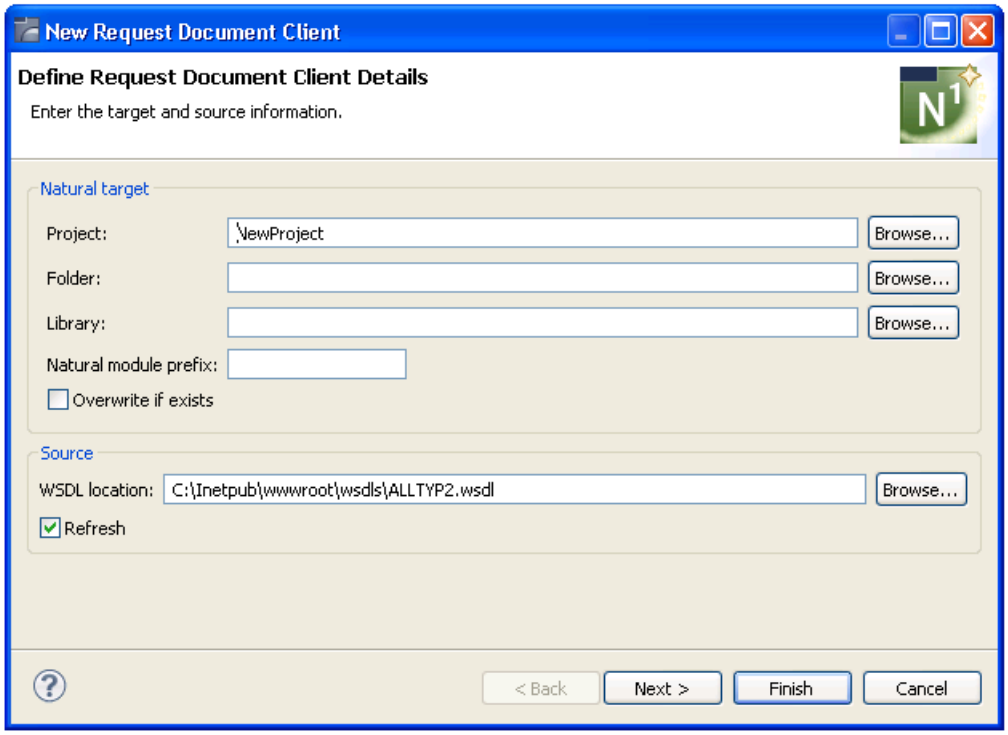
1 Open the context menu in the **Project Explorer** view for the NaturalONE project into which you want to generate the modules.

Or:

Open the context menu in the **Project Explorer** view for the library into which you want to generate the modules.

2 Select **Code Generation > New Request Document Client**.

The **Define Request Document Client Details** panel is displayed. For example:



Using this panel, you can:

Task	Procedure
Select another NaturalONE project in which to generate the REQUEST-DOCUMENT client modules.	Type the name of the project in **Project** or select **Browse** to display a window listing the existing projects for selection. The project must currently exist.
Select a folder in which to generate the REQUEST-DOCUMENT client modules.	Type the name of the folder in **Folder** or select **Browse** to display a window listing the existing folders for selection. The folder must currently exist within the selected NaturalONE project. **Note:** This option allows you to generate modules into more complex library structures (for example, "Natural-Libraries/*my library* (MYLIB)/SRC"). When this option is not specified, the modules will be generated into the basic library folder (for example, "Natural-Libraries/MYLIB/SRC", "Natural-Libraries/MYLIB/Subprograms", etc.).

Task	Procedure
Assign a prefix to the generated Natural module names.	Type the prefix in **Natural module prefix**. Several Natural modules are created during generation, such as PDAs, subprogram(s), and LDA(s). This prefix will be used as the first character in the module names to help identify them as belonging to this REQUEST-DOCUMENT client.
Replace an existing subprogram with the same name in the same library with the one you are creating.	Select **Overwrite if exists**.

3 Type the location of the Natural library in which to generate the subprogram and associated modules in **Library**.

The library must currently exist.

Or:

Select **Browse** to display a window listing the existing libraries for selection.

Note: The libraries listed for selection are based on the current project.

4 Type a valid WSDL path (HTTP location) in **WSDL location**.

Or:

Select **Browse** to display a window listing WSDL locations for selection.

The code generator will scan the selected WSDL file for each Web service operation and generate a separate subprogram for each one.

Notes:

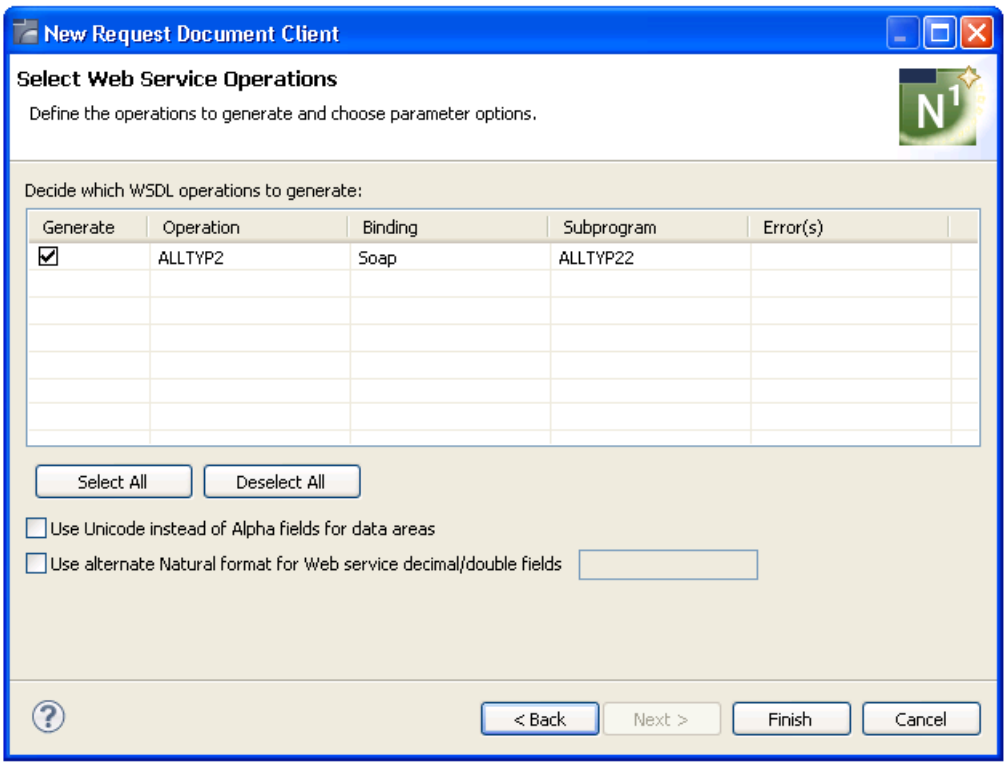
1. WSDLs that use SOAP RPC encoding (*http://schemas.xmlsoap.org/soap/encoding*) are not supported. SOAP RPC encoding does not conform to the Web Service Interoperability standards (WS-I). For more information, refer to *http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html#refinement16448072*.
2. By default, **Refresh** is selected and the code generator will retrieve the operations defined for the Web service. If you do not want the operations retrieved, deselect **Refresh**.

5 Select **Finish** to generate the REQUEST-DOCUMENT client with all default operations.

Or:

Select **Next** to select which operations to generate.

The wizard reads the specified WSDL, determines which operations it contains, and displays the **Select Web Services Operations** panel, showing the operations defined for the Web service. Each operation is represented by a line in a table. For this example, "C:\Inetpub\www-root\wsdls\ALLTYP2.wsdl" was used as the WSDL location:



The **Select Web Service Operations** panel displays the following details for each operation:

- Whether a subprogram will be generated (yes)
- Which operation will be generated (ALLTYP2)
- Which binding will be used (SOAP)
- What the generated subprogram will be named (ALLTYP21)

Using this panel, you can:

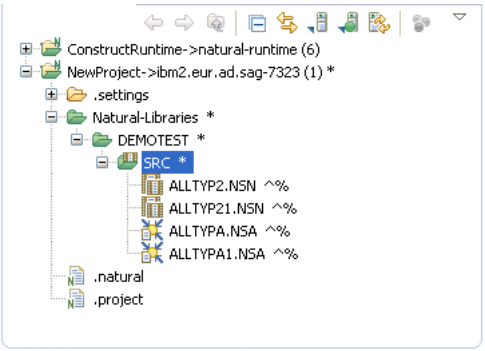
Task	Procedure
Suppress the generation of one or more operations.	Deselect the operation(s) in **Generate** and a REQUEST-DOCUMENT subprogram will not be generated for that operation. A subprogram will only be generated for each operation that is selected in **Generate**. **Note:** A minimum of one operation must be selected.

Task	Procedure
Change the type of binding used.	Select another type of binding in **Binding**. **Note:** The wizard defaults to the binding that is appropriate for the specified WSDL. We recommend that you do not change the default binding.
Change the name of the subprogram to be generated.	Type the new name in **Subprogram**.
Select all operations.	Select **Select All**. This option allows you to quickly select all operations.
Deselect all operations.	Select **Deselect All**. This option allows you to quickly deselect all operations. **Note:** A minimum of one operation must be selected.
Use Unicode format instead of alphanumeric format for variables in the data areas.	Select **Use Unicode instead of Alpha fields for data areas**. Select this option if the Web service passes Unicode data. With Natural, this is determined by whether the Natural server is configured to use Unicode variables. If the Natural server is not configured to use Unicode, do not select this option and the generator will generate a REQUEST-DOCUMENT client that contains no Unicode variables. **Note:** This option defaults to the value defined for the **Generate Unicode Dynamics** option in the **Preferences** window for **Natural**. For information, see *Set Natural Preferences*.
Generate data areas using an alternate Natural format for decimal or double Web service fields.	Select **Use alternate Natural format for Web service decimal/double fields**. Select this option if you want to generate data areas using an alternate Natural format for Web service fields of type decimal or double and then type the new format in the input field. **Note:** If this option is not selected, the default Natural format will be used (F8).

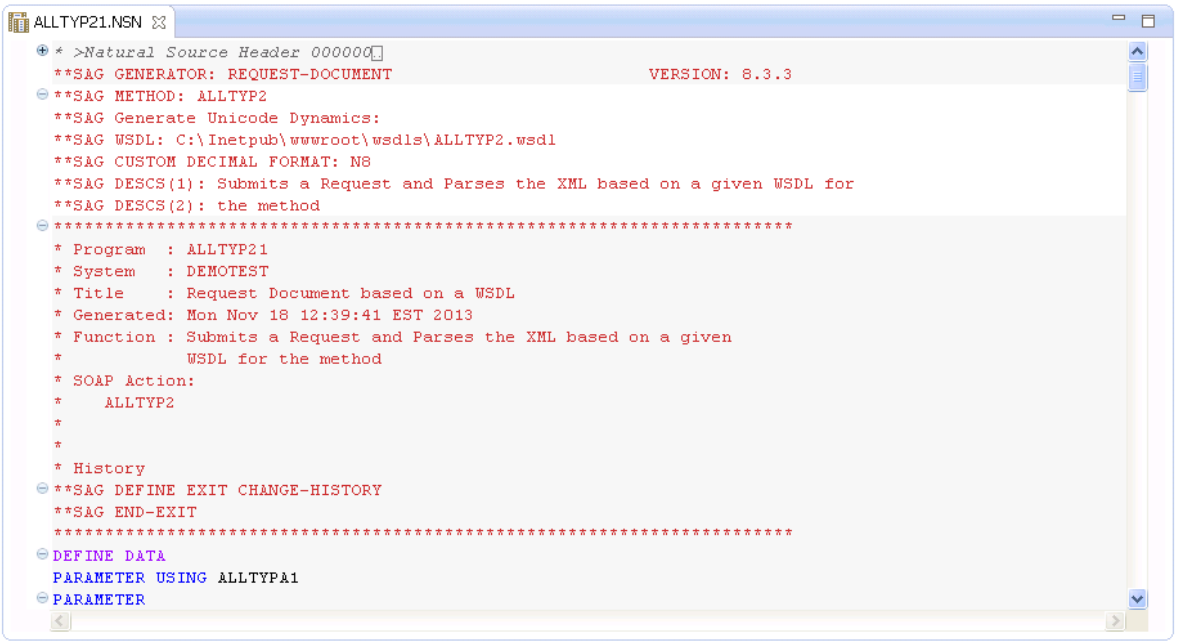
Note: If desired, a **Generation Progress** window can be displayed during generation. For information, see *Set Code Generation Preferences*.

6 Select **Finish**.

The subprogram is generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view (see *User Exits for the REQUEST-DOCUMENT Subprogram*) and the generated modules are displayed in the **Project Explorer**. For example:



The generated subprogram is displayed in the editor view. For example:



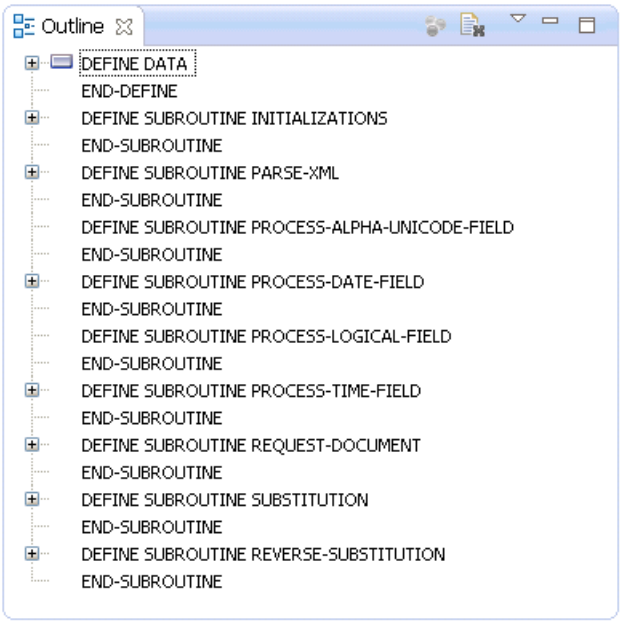
7 Save the generated module.

At this point, you can:

- Define user exits for the subprogram. For information, see ***Defining User Exits***.
- Use the NaturalONE Testing option to test the subprogram. For information, see *Test a Subprogram Directly* in *Application Testing*.
- Use NaturalONE functionality to upload the generated subprogram to the server.

User Exits for the REQUEST-DOCUMENT Subprogram

The **Outline** view for the REQUEST-DOCUMENT subprogram displays the available user exits. For example:



You can use these exits to define additional processing.

Notes:

1. All user exits are empty when generated.
2. For information about adding custom code within user exits, see *Defining User Exits*.

Define XML Substitution Characters

The generated REQUEST-DOCUMENT subprogram translates special characters (such as xml tags) in and out of the data it passes. To determine the substitutions for these characters, the REQUEST-DOCUMENT subprogram uses the CDXMLSU2 subprogram and CDRDOCA2 parameter data area (PDA) in the SYSTEM library. The REQUEST-DOCUMENT subprogram calls CDXMLSU2 to set up the XML substitution characters. Both of these modules are shipped with the Construct runtime project.

Note: For information about adding this project, see *Add the Construct Runtime Project*.

The following example shows CDXMLSU2 in the editor view:

```
CDXMLSU2.NSN
* >Natural Source Header 000000
**SAG GENERATOR: CST-SHELL                          VERSION: 5.3.1.10
**SAG TITLE: CST XML substitution mod
**SAG SYSTEM: C53RT
**SAG DESCS(1): This CST module is used to substitute XML reserved
**SAG DESCS(2): characters found in the xml data
**SAG PROGRAM-TYPE: N
**SAG PARAMETER-NAME(1): CDRDOCA2
************************************************************************
* Program  : CDXMLSUB
* System   : C53RT
* Title    : CST XML substitution mod
* Generated: May 17,2011 at 09:59:05 by SAG
* Function : This CST module is used to substitute XML reserved
*            characters found in the xml data
*
*
* History
**SAG DEFINE EXIT CHANGE-HISTORY
* Changed on May 13,2011 by CNDSHE for release ____
* >
* >
* >
**SAG END-EXIT
************************************************************************
DEFINE DATA
**SAG EXIT POINT PARAMETER-DATA
  PARAMETER USING CDRDOCA2
  LOCAL USING CSASTD
  LOCAL USING CSLRCODE                /* Message return codes
  LOCAL
  01 #PROGRAM (A8)
**SAG DEFINE EXIT LOCAL-DATA
  LOCAL
  01 #DOUBLE-QUOTE (A1)
  01 #MAX-REPLACEMENTS (I2) CONST <5> /* CUSTOMIZE: adjust table size
**SAG END-EXIT
END-DEFINE
PROG.    /* to allow escape from routine.
```

Tip: Within the editor, you can quickly find locations that must be changed by searching for "/* CUSTOMIZE".

To change settings for the XML substitution characters, use the GENERATE-CODE user exit. For example:

```
CDXMLSU2.NSN
**SAG DEFINE EXIT GENERATE-CODE
/** New Subprogram CDXMLSUB.
RESIZE ARRAY #REPLACEMENT-TABLE TO (1:#MAX-REPLACEMENTS)
* Determine double quotes in based on platform
#DOUBLE-QUOTE := 'A'
IF #DOUBLE-QUOTE < '0' THEN   /* Letters LT numbers
#DOUBLE-QUOTE := H'7F' /* EBCDIC
ELSE
#DOUBLE-QUOTE := H'22' /* ASCII
END-IF
*
* setup search and replace strings
* note & must be first because of & substitution
/* CUSTOMIZE:
#SEARCH-STRING(1) := '&'
#REPLACE-STRING(1) := '&amp;'
#SEARCH-STRING(2) := "'"
#REPLACE-STRING(2) := '&apos;'
#SEARCH-STRING(3) := #DOUBLE-QUOTE
#REPLACE-STRING(3) := '&quot;'
#SEARCH-STRING(4) := '<'
#REPLACE-STRING(4) := '&lt;'
#SEARCH-STRING(5) := '>'
#REPLACE-STRING(5) := '&gt;'
**SAG END-EXIT
*
```

In this example, the LOCAL-DATA user exit defines the Natural format for the #DOUBLE-QUOTE and #MAX-REPLACEMENTS values and the size of the #REPLACEMENT-TABLE array. The GENERATE-CODE user exit resizes the #REPLACEMENT-TABLE array and defines the logic and substitution values for #DOUBLE-QUOTE.

This section covers the following topics:

- Add XML Substitution Characters
- Modify XML Substitution Characters
- Delete XML Substitution Characters

Add XML Substitution Characters

To add xml substitution characters

1. Select and open CDXMLSU2 in the Construct runtime project.
2. Increase the size of the #MAX-REPLACEMENTS value for the #REPLACEMENT-TABLE array by "*n*" in the LOCAL-DATA user exit, where "*n*" is the number of substitution characters you are adding.
3. Assign the #SEARCH-STRING and #REPLACE-STRING values and indexes for each substitution character you are adding.
4. Stow the CDXMLSU2 subprogram in the SYSTEM library.

Modify XML Substitution Characters

To modify xml substitution characters

1 Select and open CDXMLSU2 in the Construct runtime project.

2 Change the #SEARCH-STRING and #REPLACE-STRING values and indexes for each substitution character you are modifying.

3 Stow the CDXMLSU2 subprogram in the SYSTEM library.

Delete XML Substitution Characters

To delete xml substitution characters

1 Select and open CDXMLSU2 in the Construct runtime project.

2 Decrease the size of the #MAX-REPLACEMENTS value for the #REPLACEMENT-TABLE array by "*n*" in the LOCAL-DATA user exit, where "*n*" is the number of substitution characters you are deleting.

3 Delete the #SEARCH-STRING and #REPLACE-STRING values and indexes for each substitution character you are deleting.

4 Stow the CDXMLSU2 subprogram in the SYSTEM library.

4 Create an Object-Maintenance Process

- Generate the Object Maint Subprogram 24
- User Exits for the Object Maint Subprogram 30

This section describes the Object Maint code generator, which creates a subprogram that maintains complex data objects and updates all entities within an object. The generator also creates the local and parameter data areas.

Generate the Object Maint Subprogram

To generate an object-maintenance subprogram and data areas

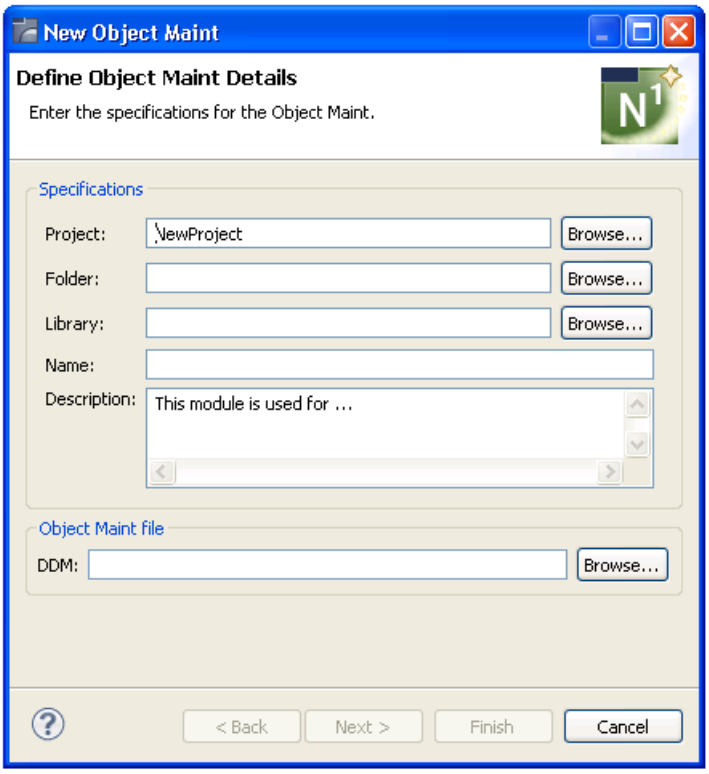
1 Open the context menu in the **Project Explorer** view for the NaturalONE project into which you want to generate the modules.

Or:

Open the context menu in the **Project Explorer** view for the library into which you want to generate the modules.

2 Select **Code Generation > New Object Maint**.

The **Define Object Maint Details** panel is displayed. For example:



3 Type the location of the Natural library in which to generate the subprogram and associated modules in **Library**.

The library must currently exist.

Or:

Select **Browse** to display a window listing the existing libraries for selection.

Note: The libraries listed for selection are based on the current project.

4 Type the name of the object maint subprogram in **Name**.

5 Select the DDM for the object maint subprogram in **DDM**.

Tip: The DDMs are typically located in the SYSTEM library.

Using this panel, you can:

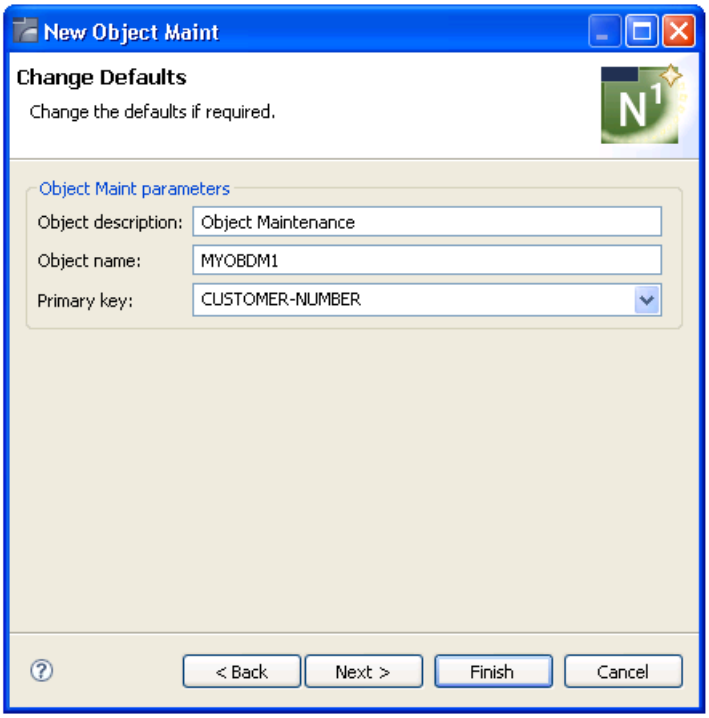
Task	Procedure
Select another NaturalONE project in which to generate the object maint modules.	Type the name of the project in **Project** or select **Browse** to display a window listing the existing projects for selection. The project must currently exist.
Select a folder in which to generate the object-maintenance modules.	Type the name of the folder in **Folder** or select **Browse** to display a window listing the existing folders for selection. The folder must currently exist within the selected NaturalONE project. **Note:** This option allows you to generate modules into more complex library structures (for example, "Natural-Libraries/*my library* (MYLIB)/SRC"). When this option is not specified, the modules will be generated into the basic library folder (for example, "Natural-Libraries/MYLIB/SRC", "Natural-Libraries/MYLIB/Subprograms", etc.).
Provide a description of the object maint subprogram.	Type a brief description in **Description**.

6 Select **Finish** to generate the object maint subprogram and associated modules with the default values.

Or:

Select **Next** to change the default specification values.

The **Change Defaults** panel is displayed. For example:



This panel displays the default specification values for the subprogram to be generated. Using this panel, you can:

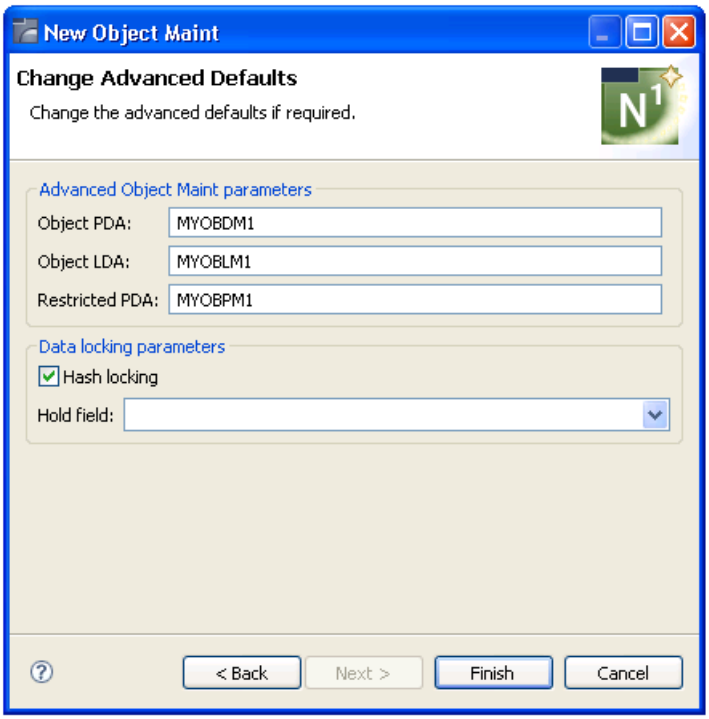
Task	Procedure
Change the description of the subprogram to be generated.	Type the description in **Object description**.
Change the name of the object.	Type the name in **Object name**.
Change the primary key field used for maintenance operations.	Select the field in **Primary key**.

7 Select **Finish** to generate the object maint subprogram and associated modules.

Or:

Select **Next** to change the default parameter values.

The **Change Advanced Defaults** panel is displayed. For example:



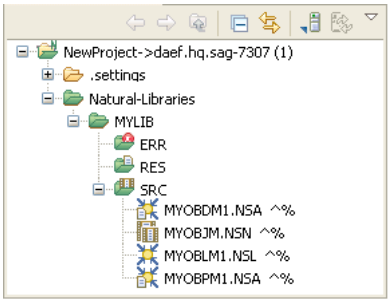
This panel displays the default parameter values for the subprogram to be generated. Using this panel, you can:

Task	Procedure
Change the name of the parameter data area (PDA) for the object.	Type the name in **Object PDA**.
Change the name of the local data area (LDA) for the object.	Type the name in **Object LDA**. **Note:** The local data area is only required when the hash-locking option is used for record locking.
Change the name of the restricted parameter data area (PDA) for the object.	Type the name in **Restricted PDA**.
Use a hold field to lock data for maintenance operations.	Select the hold field in **Hold field**. For more information, see *Record-Locking Options*. **Note:** By default, the hash-locking mechanism is used to lock data. If you select a hold field, the **Hash locking** field is automatically deselected.

Note: If desired, a **Generation Progress** window can be displayed during generation. For information, see *Set Code Generation Preferences*.

8 Select **Finish**.

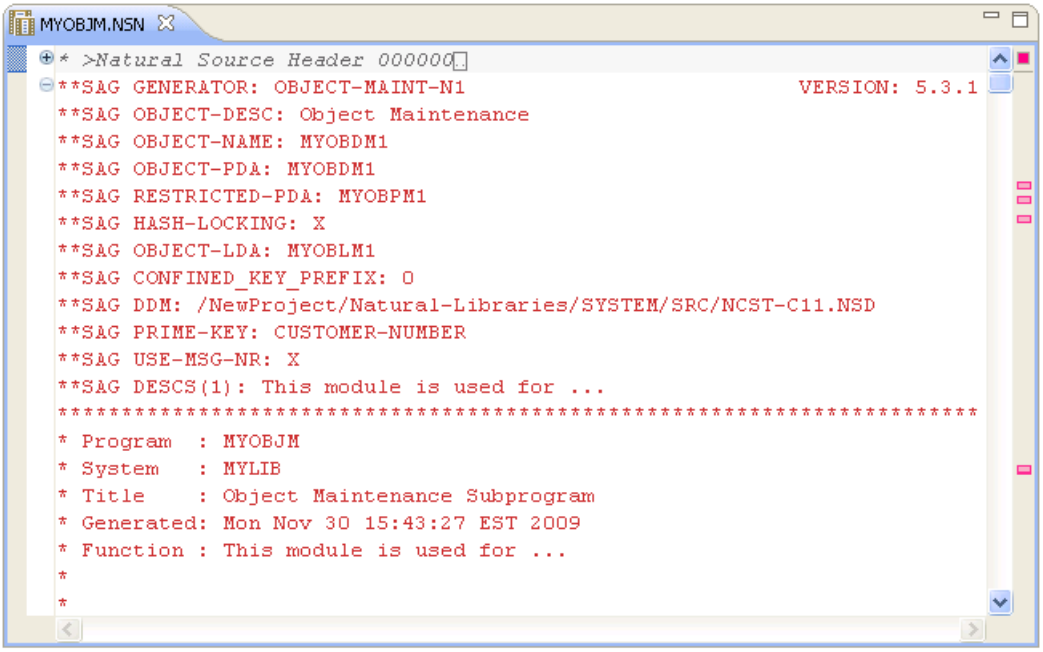
The subprogram is generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view (see *User Exits for the Object Maint Subprogram*) and the generated modules are displayed in the **Project Explorer**. For example:



These modules are:

Module	Description
MYOBDM1.NSA	Parameter data area for the object maint subprogram
MYOBJM.NSN	Object maint subprogram
MYOBLM1.NSL	Local data area for the object maint subprogram
MYOBPM1.NSA	Restricted parameter data area for the object maint subprogram

The subprogram is displayed in the editor view. For example:



9 Save the subprogram and associated modules.

At this point, you can:

- Define user exits for the subprogram. For information, see ***Defining User Exits***.
- Use the NaturalONE Testing option to test the subprogram. For information, see *Test a Subprogram Directly* in *Application Testing*.
- Use NaturalONE functionality to upload the generated subprogram to the server.

Record-Locking Options

In a client/server environment, data retrieved for maintenance is not locked initially. Instead, the object maint subprogram retrieves the data again and locks it prior to updating, storing, or deleting data from the database.

To ensure changes are not overwritten by another user during this process, the subprogram must determine whether the data has changed since the initial retrieval. To do this, the object maint subprogram has two record-locking options:

- Hash-locking

  This method is the most reliable. The subprogram retrieves the initial data and hashes it to a number. When it retrieves the data to lock it, the subprogram hashes it to a number again. Logical variables are stored in alphanumeric format in the local data area to process the hashed values. All data must hash to the same value as when it was requested. If it does, data has not changed and the changes are allowed.

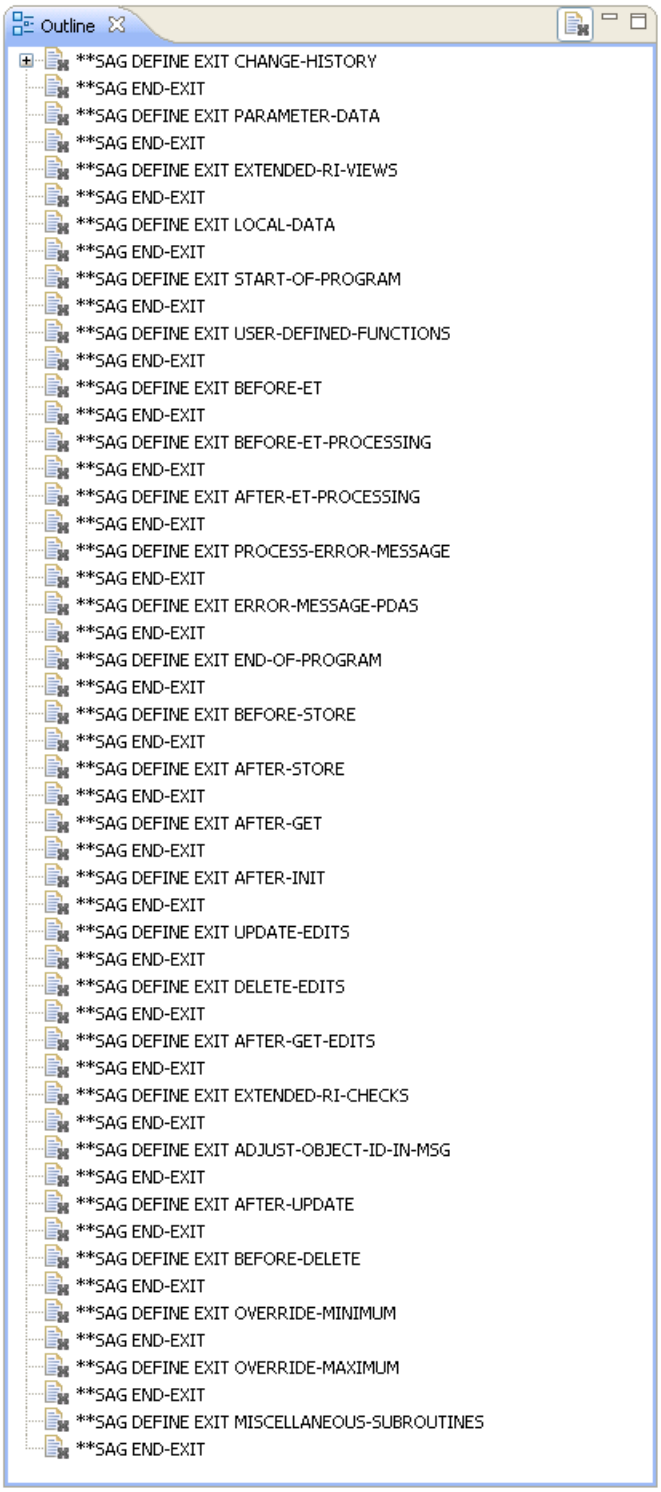
- Timestamp

  The timestamp (or counter) method is the traditional record-locking mechanism. This method assumes that every time data changes, the timestamp also changes. This method is more efficient than the hash-locking method because the subprogram only has to check one field, but this assumption can be incorrect when the file is not maintained by an Object Maint-generated subprogram (for example, a programmer-coded subprogram may not change the timestamp when data is modified).

If the file is not normally maintained through an Object Maint-generated subprogram, the hash-locking option should be used. If the file is only maintained through an Object Maint-generated subprogram, the timestamp option should be used (as it is more efficient). For more information, see *Natural Construct Object Models*.

User Exits for the Object Maint Subprogram

The **Outline** view for the object-maintenance subprogram displays the available user exits. For example:



You can use these exits to define additional processing.

Notes:

1. All user exits are empty when generated.
2. For information about adding custom code within user exits, see ***Defining User Exits***.

5 Create an Object-Browse Process

- Introduction to the Object-Browse Code Generator 34
- Generate the Object-Browse Subprogram 34
- User Exits for the Object-Browse Subprogram 38

This section describes the Object-Browse code generator.

Introduction to the Object-Browse Code Generator

The Object-Browse code generator creates the browse subprogram for an object, as well as three parameter data areas:

Data Area	Description
Object PDA	Defines the returned row data.
Object key PDA	Defines the search key values.
Restricted PDA	Contains private data used internally by the browse object to maintain context.

Generate the Object-Browse Subprogram

To generate an object-browse subprogram and data areas

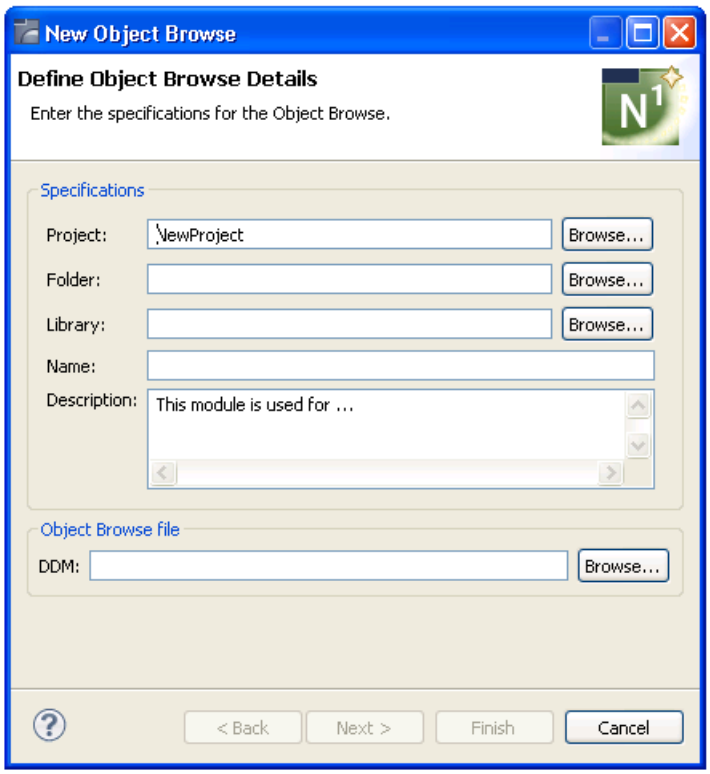
1 Open the context menu in the **Project Explorer** view for the NaturalONE project into which you want to generate the modules.

   Or:

   Open the context menu in the **Project Explorer** view for the library into which you want to generate the modules.

2 Select **Code Generation > New Object Browse**.

   The **Define Object Browse Details** panel is displayed. For example:



3 Type the location of the Natural library in which to generate the subprogram and associated modules in **Library**.

The library must currently exist.

Or:

Select **Browse** to display a window listing the existing libraries for selection.

Note: The libraries listed for selection are based on the current project.

4 Type the name of the object-browse subprogram in **Name**.

5 Select the DDM for the object-browse subprogram in **DDM**.

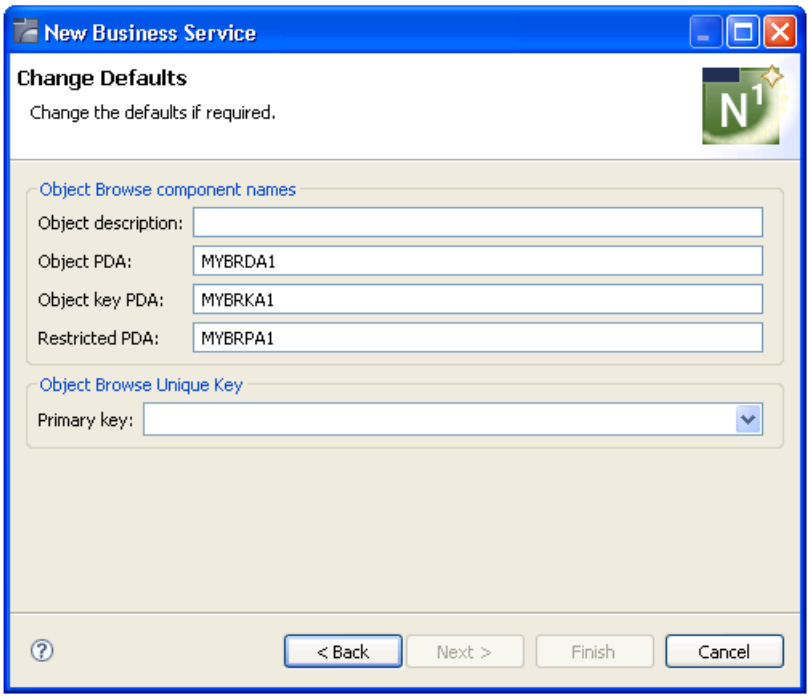
Tip: The DDMs are typically located in the SYSTEM library.

Using this panel, you can:

Task	Procedure
Select another NaturalONE project in which to generate the object-browse modules.	Type the name of the project in **Project** or select **Browse** to display a window listing the existing projects for selection. The project must currently exist.
Select a folder in which to generate the object-browse modules.	Type the name of the folder in **Folder** or select **Browse** to display a window listing the existing folders for selection. The folder must currently exist within the selected NaturalONE project. **Note:** This option allows you to generate modules into more complex library structures (for example, "Natural-Libraries/*my library* (MYLIB)/SRC"). When this option is not specified, the modules will be generated into the basic library folder (for example, "Natural-Libraries/MYLIB/SRC", "Natural-Libraries/MYLIB/Subprograms", etc.).
Change or provide a description of the object-browse subprogram.	Type a brief description in **Description**.

6 Select **Next**.

The **Change Defaults** panel is displayed. For example:

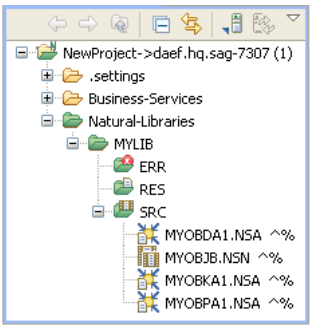


This panel displays the default specification values for the subprogram to be generated. Using this panel, you can:

Task	Procedure
Provide a description of the subprogram to be generated.	Type the description in **Object description**.
Change the name of the object PDA.	Type the name in **Object PDA**.
Change the name of the object key PDA.	Type the name in **Object key PDA**.
Change the name of the restricted PDA.	Type the name in **Restricted PDA**.
Define the primary key field used for browse operations.	Select the field in **Primary key**. **Note:** This option is only available when the primary key is not known (for example, DB2 files). For Adabas files, the primary key is the ISN.

7 Select **Finish**.

When generation is complete, the available user exits are displayed in the **Outline** view (see *User Exits for the Object-Browse Subprogram*) and the generated modules are displayed in the **Project Explorer**. For example:



These modules are:

Module	Description
MYOBDA1.NSA	Object PDA
MYOBJB.NSN	Object-browse subprogram
MYOBKA1.NSA	Object key PDA
MYOBPA1.NSA	Restricted PDA

The subprogram is displayed in the editor view. For example:

```
MYOBJB.NSN
* >Natural Source Header 000000
**SAG GENERATOR: OBJECT-BROWSE-N1                          VERSION: 5.3.1
**SAG OBJECT-DESC: Object Browse
**SAG OBJECT-NAME: MYOBJB
**SAG ROW-PDA-NAME: MYOBDA1
**SAG DDM: /NewProject/Natural-Libraries/SYSTEM/SRC/NCST-CUS.NSD
**SAG USE-MSG-NR: X
**SAG KEY-PDA-NAME: MYOBKA1
**SAG RESTRICTED-PDA-NAME: MYOBPA1
**SAG DESCS(1): This module is used for ...
**SAG HISTOGRAM(1): X
**SAG PHYSICAL-KEY(1,1): CUSTOMER-NUMBER
**SAG HISTOGRAM(2): X
**SAG PHYSICAL-KEY(2,1): BUSINESS-NAME
**SAG HISTOGRAM(3): X
**SAG PHYSICAL-KEY(3,1): CUSTOMER-WAREHOUSE-ID
**SAG HISTOGRAM(4): X
**SAG PHYSICAL-KEY(4,1): CUSTOMER-TIMESTAMP
************************************************************************
* Program  : MYOBJB
* System   : MYLIB
* Title    : Object Browse Subprogram
* Generated: Tue Dec 08 16:24:03 EST 2009
* Function : This module is used for ...
```

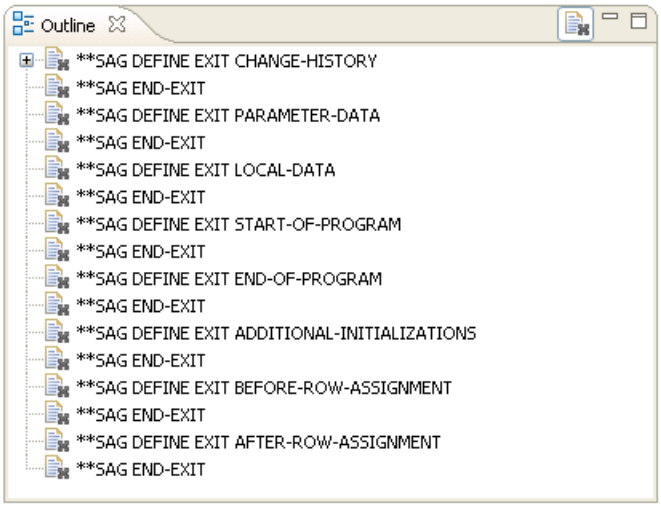
8 Save the subprogram and associated modules.

At this point, you can:

- Define user exits for the subprogram. For information, see ***Defining User Exits***.
- Use the NaturalONE Testing option to test the subprogram. For information, see *Test a Subprogram Directly* in *Application Testing*.
- Use NaturalONE functionality to upload the generated subprogram to the server.

User Exits for the Object-Browse Subprogram

The **Outline** view for the object-browse subprogram displays the available user exits. For example:



You can use these exits to define additional processing.

Notes:

1. All user exits are empty when generated.
2. For information about adding custom code within user exits, see *Defining User Exits*.

6 Create an Object Skeleton Subprogram

- Generate the Object Skeleton Subprogram 42
- User Exits for the Object Skeleton Subprogram 47

This section describes the Object Skeleton code generator, which creates a skeleton subprogram.

Generate the Object Skeleton Subprogram

To generate an object skeleton subprogram

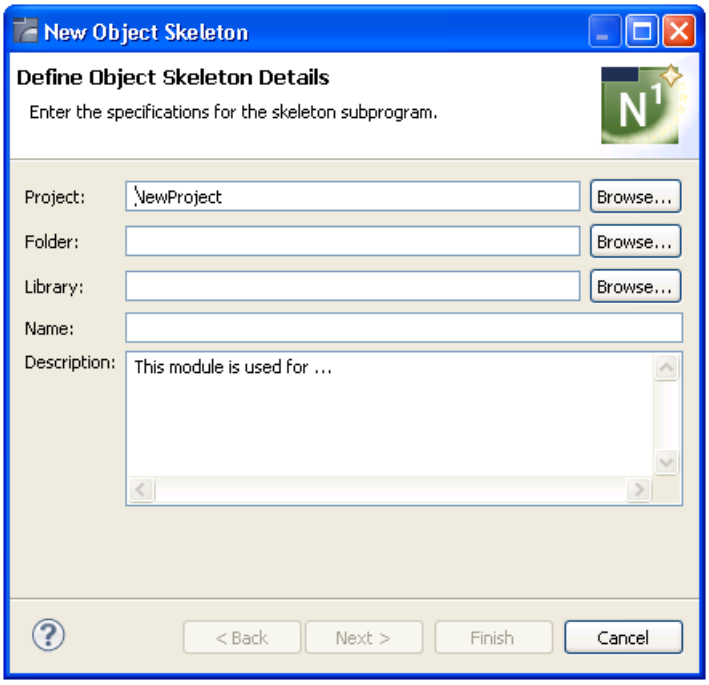
1 Open the context menu in the **Project Explorer** view for the NaturalONE project into which you want to generate the module.

Or:

Open the context menu in the **Project Explorer** view for the library into which you want to generate the module.

2 Select **Code Generation > New Object Skeleton**.

The **Define Object Skeleton Details** panel is displayed. For example:



3 Type the location of the Natural library in which to generate the subprogram and associated modules in **Library**.

The library must currently exist.

Or:

Select **Browse** to display a window listing the existing libraries for selection.

Note: The libraries listed for selection are based on the current project.

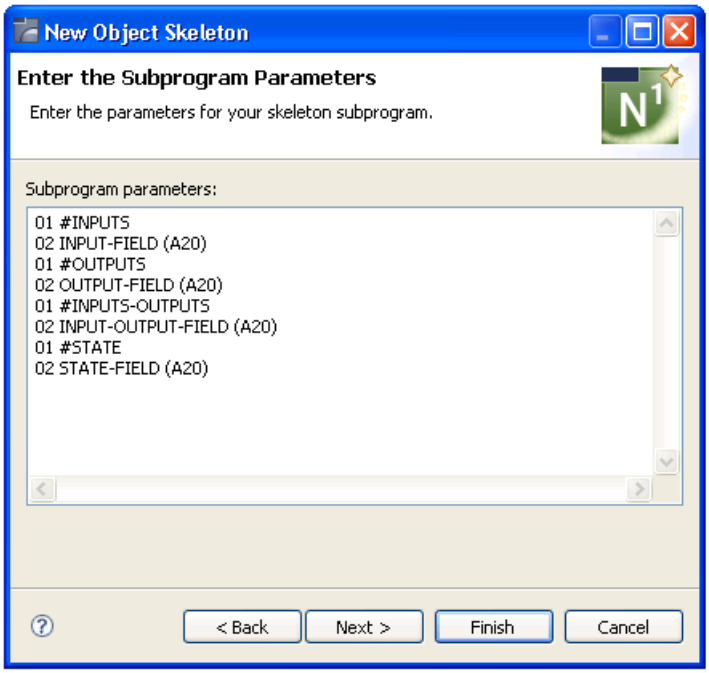
4 Type the name of the subprogram in **Name**.

Using this panel, you can:

Task	Procedure
Select another NaturalONE project in which to generate the subprogram.	Type the name of the project in **Project** or select **Browse** to display a window listing the existing projects for selection. The project must currently exist.
Select a folder in which to generate the subprogram.	Type the name of the folder in **Folder** or select **Browse** to display a window listing the existing folders for selection. The folder must currently exist within the selected NaturalONE project. **Note:** This option allows you to generate modules into more complex library structures (for example, "Natural-Libraries/*my library* (MYLIB)/SRC"). When this option is not specified, modules will be generated into the basic library folder (for example, "Natural-Libraries/MYLIB/SRC", "Natural-Libraries/MYLIB/Subprograms", etc.).
Change or provide a description of the subprogram.	Type a brief description in **Description**.

5 Select **Next**.

The **Enter the Subprogram Parameters** panel is displayed. For example:



Use this panel to define input/output and state parameters for the subprogram to be generated.

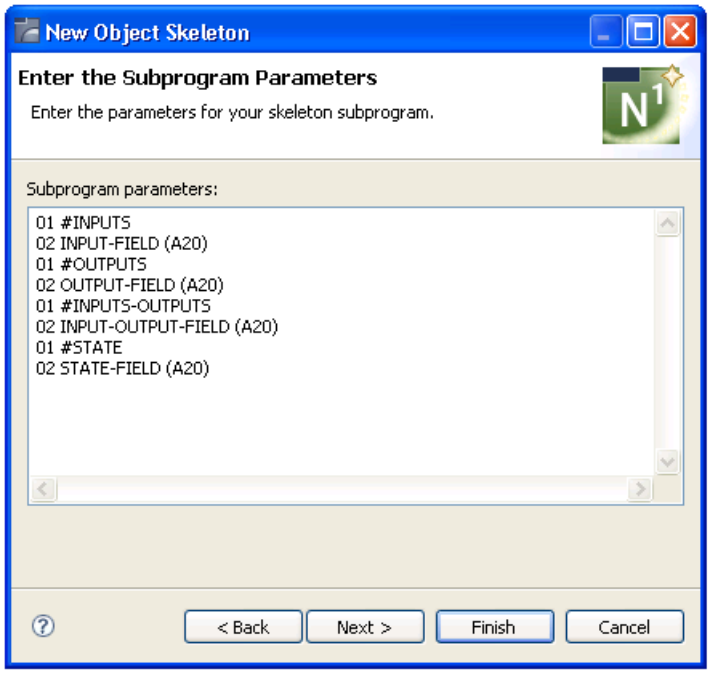
6 Type the parameters for your subprogram in **Subprogram parameters**.

7 Select **Finish** to generate the subprogram with the DEFAULT method.

Or:

Select **Next** to define other methods.

The **Enter the Subprogram Methods** panel is displayed. For example:

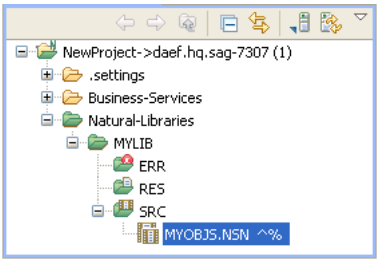


This panel displays the default methods for the subprogram to be generated. Using this panel, you can:

Task	Procedure
Add a method to the subprogram.	For information, see *Add a Method*.
Remove a method from the subprogram.	Select the method in the **Method** list and select **Remove**.

8 Select **Finish**.

When generation is complete, the available user exits are displayed in the **Outline** view (see *User Exits for the Object Skeleton Subprogram.*) and the generated modules are displayed in the **Project Explorer**. For example:



The subprogram is displayed in the editor view. For example:

```
MYOBJS.NSN
* >Natural Source Header 000000
**SAG GENERATOR: OBJECT-SKELETON-N1                    VERSION: 8.1.2
**SAG DESCS(1): This module is used for ...
**SAG METHOD-NAME(1): DEFAULT
************************************************************************
* Program  : MYOBJS
* System   : MYLIB
* Title    : Object Skeleton Subprogram
* Generated: Mon Nov 08 13:09:58 EST 2010
* Function : This module is used for ...
*
*
*
* History
**SAG DEFINE EXIT CHANGE-HISTORY
**SAG END-EXIT
************************************************************************
DEFINE DATA PARAMETER
**SAG DEFINE EXIT PARAMETERS
01 #INPUTS
02 INPUT-FIELD (A20)
01 #OUTPUTS
```

9 Save the subprogram.

At this point, you can:

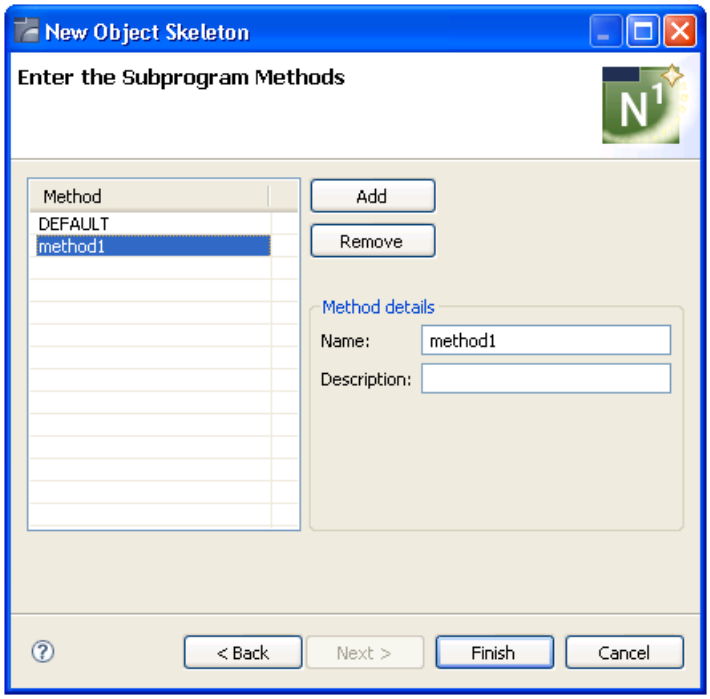
- Define user exits for the subprogram. For information, see *Defining User Exits*.
- Use the NaturalONE Testing option to test the subprogram. For information, see *Test a Subprogram Directly* in *Application Testing*.
- Use NaturalONE functionality to upload the generated subprogram to the server.

Add a Method

To add a method to the subprogram

1 Select **Add**.

The **Method details** section is displayed. For example:



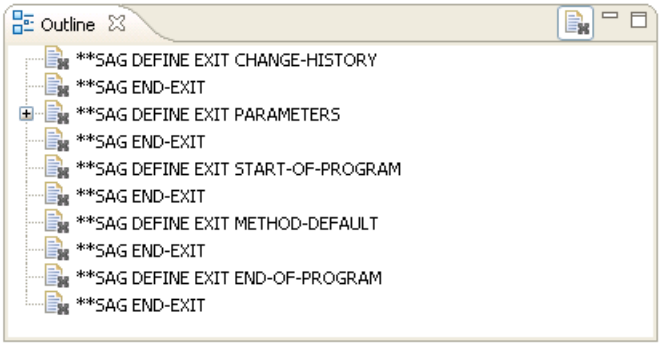
By default, method1 is displayed in the **Method** section.

2 Type the name of the new method in **Name**.

3 Type a brief description of the method in **Description**.

User Exits for the Object Skeleton Subprogram

The **Outline** view for the object skeleton subprogram displays the available user exits. For example:



You can use these exits to define additional processing.

Notes:

1. All user exits are empty when generated.
2. For information about adding custom code within user exits, see ***Defining User Exits***.

7 Regenerate Subprograms and Associated Modules

- Regenerate a Subprogram and Associated Modules 50
- Regenerate Multiple Subprograms 51
- Compare Differences 53

You can regenerate any subprogram that was generated using a supplied code generator, as well as all data areas, associated subprograms, and user exits that were generated with the subprogram. You can also select more than one project, folder, or object to regenerate multiple modules.

Regenerate a Subprogram and Associated Modules

There are two methods of regenerating a subprogram, which are represented by two context menu options:

- Regenerate

  The selected subprogram and all associated modules are regenerated without showing the generator panels.

- Regeneration using Wizard

  The first wizard panel is displayed. You can edit the settings and select **Finish** on the last panel to regenerate the selected subprogram and all associated modules.

To regenerate a subprogram and associated modules

1 Open the context menu for the subprogram in the **Project Explorer**.

2 Select **Code Generation > Regenerate**.

  The selected subprogram and all associated modules are regenerated without displaying the wizard panel(s).

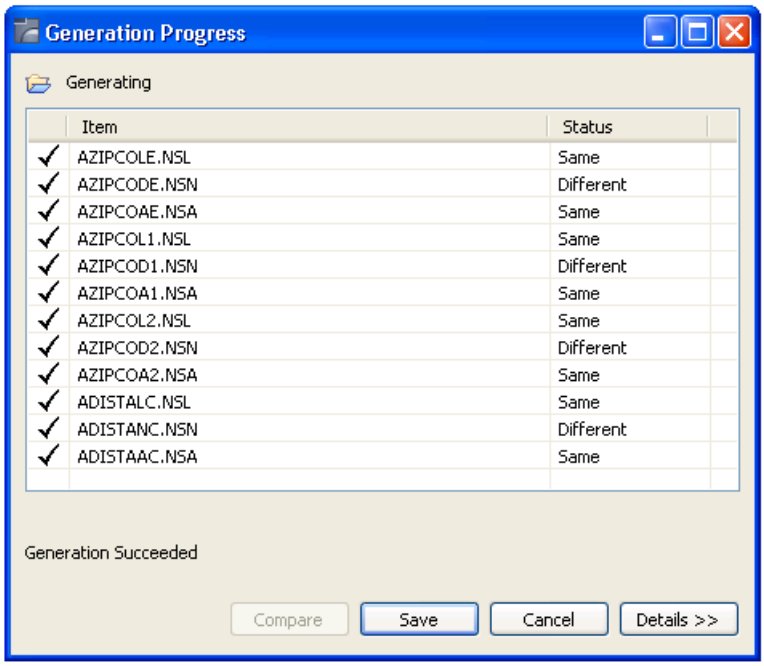
  Or:

  Select **Code Generation > Regenerate Using Wizard**.

  The code generator reads the subprogram specifications and displays the wizard panels, which are the same as those displayed when the subprogram was first generated.

  Note: For information on the specification panels, see the section describing that code generator.

  After selecting **Finish**, the **Generation Progress** window is displayed, indicating the results of the regeneration. For example:



If any of the Natural modules have changed since the subprogram was first generated, the compare option is enabled when you select the module. For information, see *Compare Differences*.

Note: The **Generation Progress** window is only displayed when the option is set in the **Preferences** window. For information, see *Set Code Generation Preferences*.

3 Select **Save** to save the regenerated subprogram and associated modules.

You can now upload all modules to the server using standard NaturalONE functionality.

Regenerate Multiple Subprograms

This section describes how to regenerate more than one subprogram and associated modules.

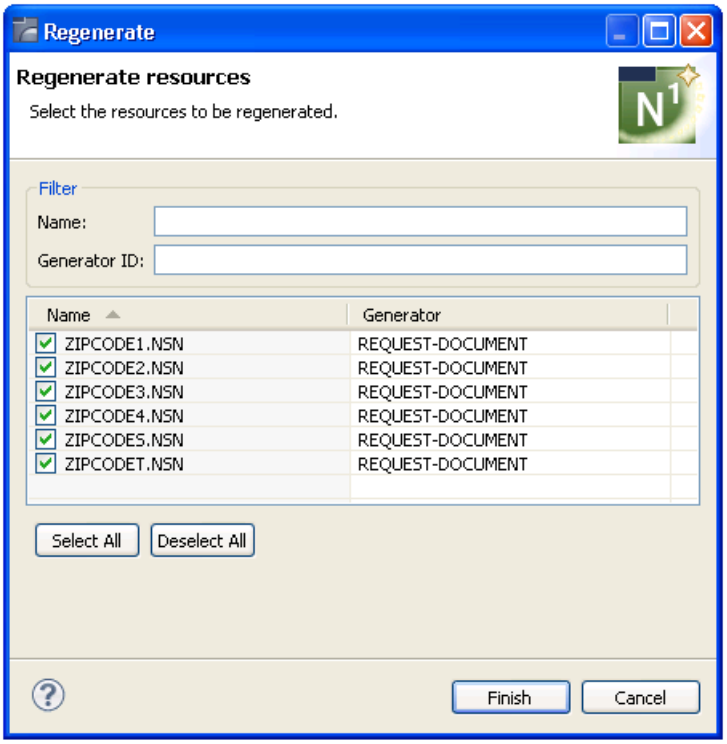
To regenerate multiple subprograms

1 Open the context menu for the subprograms in the **Project Explorer**.

You can select one or more projects, libraries, or individual subprograms using standard selection techniques.

2 Select **Regenerate**.

First, a progress window is displayed as the wizard locates and loads the regeneratable objects. Next, a selection window is displayed to choose the objects you want to regenerate. For example:

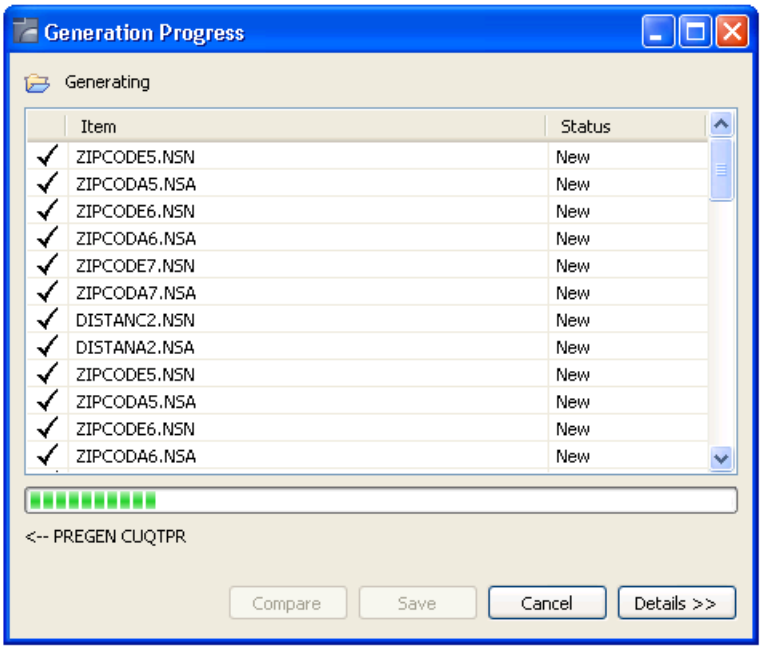


Using this panel, you can:

Task	Procedure
Filter the list of resources for selection.	Type a prefix in **Filter**. For example, if you type "ZIP", only the resources beginning with ZIP are selected.
Use a different code generator to regenerate the resource.	Type the generator ID in **Generator ID**.
Deselect all resources.	Select **Deselect All**.

3 Select **Finish**.

The **Generation Progress** window is displayed, showing the progress of the generation process. For example:



Compare Differences

The **Generation Progress** window displays the results of regeneration. If the generated modules have changed since the previous generation, Different is displayed in the **Status** column and you can display a window in which you can compare the regenerated code with the original.

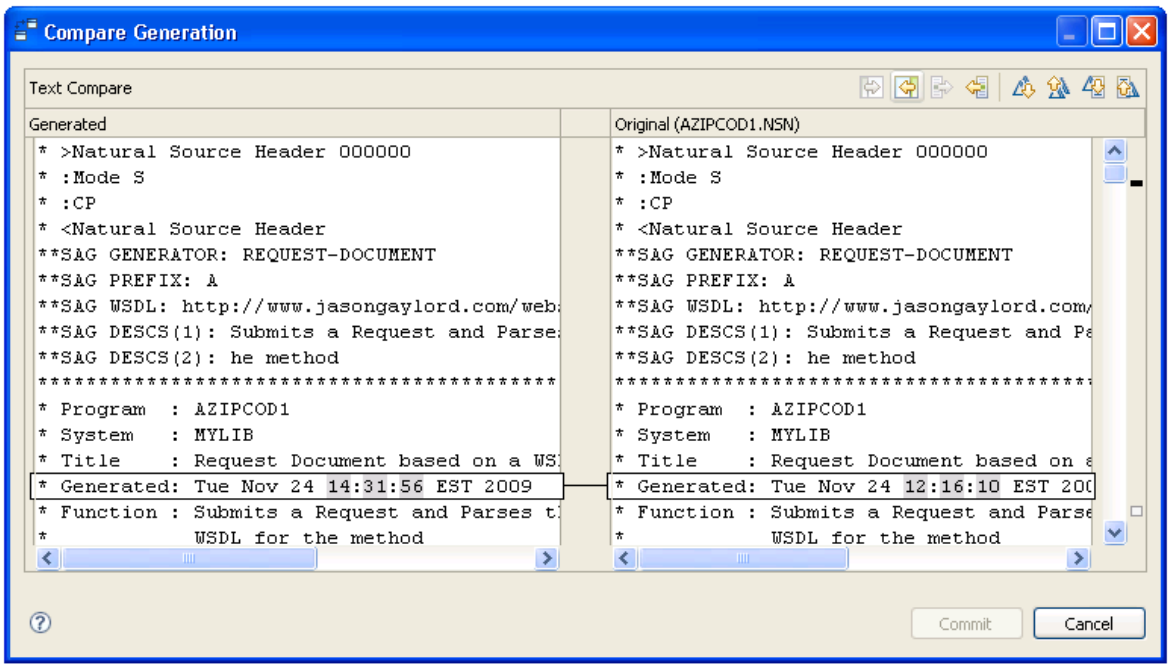
Note: You cannot compare two modules that are identical.

To compare the regeneration differences

1 Select the module for which you want to compare the differences.

2 Select **Compare**.

The **Compare Generation** window is displayed. For example:



This window displays the results of the new (Generated) and previous (Original) generation and indicates the differences.

3 Decide what to do about the differences.

4 Select **Commit** to save the changes.

8 Set Preferences

- Set Code Generation Preferences 56
- Set Logging Preferences 57
- Set Natural Preferences 59

This section describes how to set preferences for the Code Generation component supplied with NaturalONE.

Set Code Generation Preferences

This section describes how to set common generation preferences for Code Generation.

To set Code Generation preferences

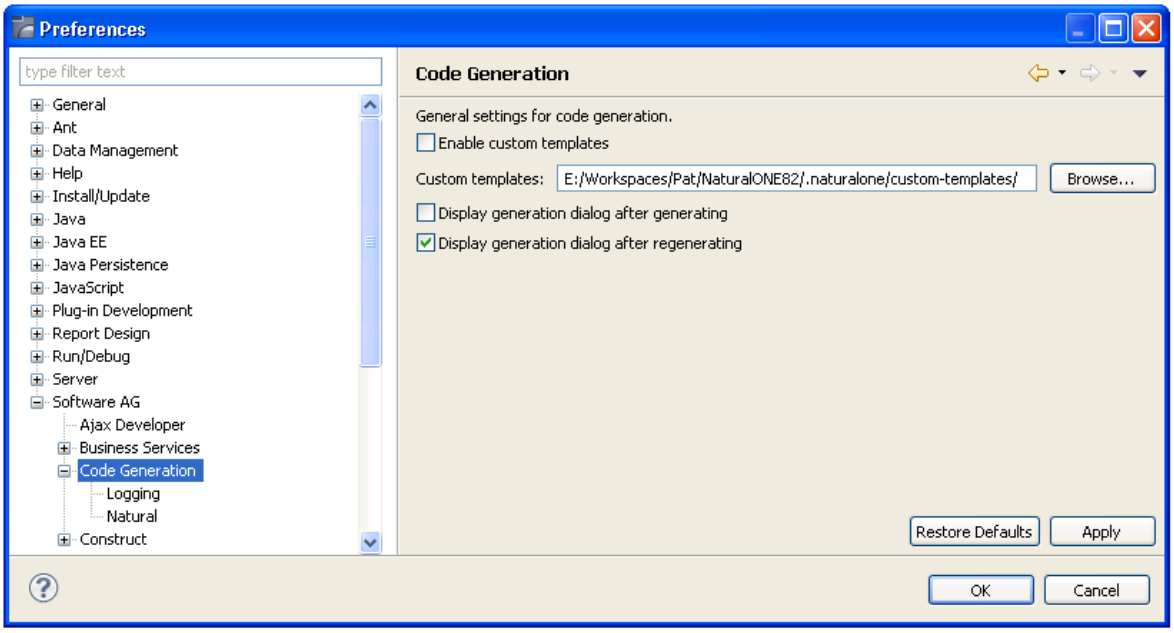
1 Select **Preferences** on the **Window** menu.

The **Preferences** window is displayed.

2 Expand the **Software AG** root node.

3 Select **Code Generation**.

The **Code Generation** preferences are displayed. For example:



Using this window, you can:

Task	Procedure
Enable customized templates and select the folder containing the templates.	Select **Enable custom templates** and use **Browse** to select the root folder for custom templates (by default, the custom-templates folder). **Note:** Do not change the underlying folder structure for the root folder or the code generator will not be able to find the custom templates.
Display the **Generation Progress** window after generation.	Select **Display generation dialog after generating**.
Not display the **Generation Progress** window after regeneration.	Deselect **Display generation dialog after regenerating**.

4 Select **OK** to save the preferences.

Set Logging Preferences

This section describes how to set preferences for logging.

To set logging preferences

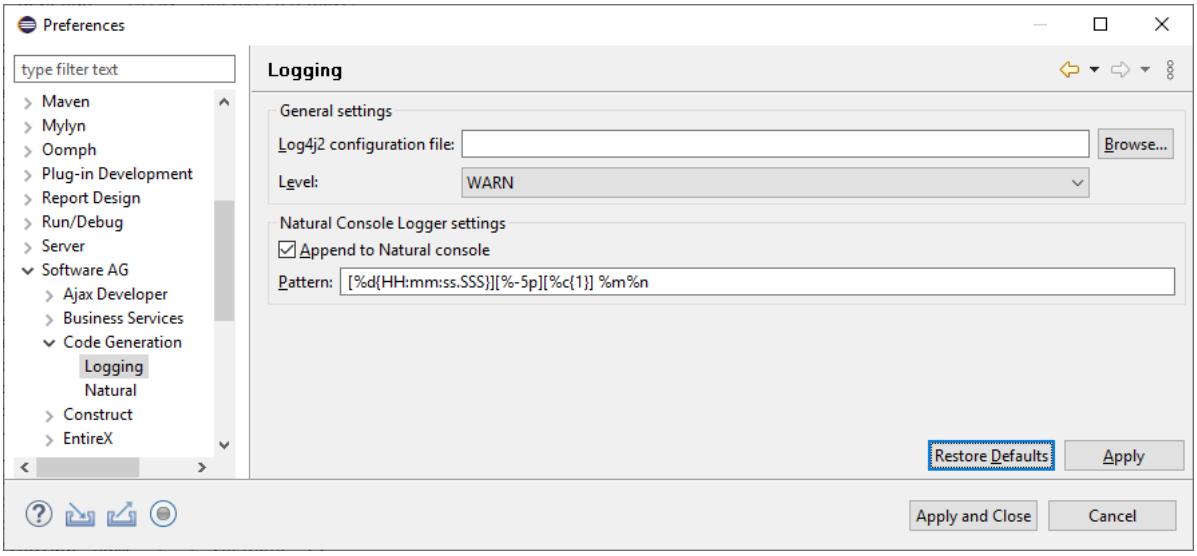
1 Select **Preferences** on the **Window** menu.

The **Preferences** window is displayed.

2 Expand the **Software AG** root node.

3 Select **Code Generation > Logging**.

The **Logging** preferences are displayed. For example:



Using this window, you can:

Task	Procedure
Assign log4j2 configuration file to use for logging.	Select **Browse** and search for the log4j 2 configuration file, e.g. *log4j2.xml*.
Change the log level.	Select the logger level in **Level**. The logger levels are: ■ OFF ■ FATAL ■ ERROR ■ WARN ■ INFO ■ DEBUG ■ TRACE ■ ALL See the Apache Log4j 2 documentation for more information.
Not append the Code Generation console view to the NaturalONE **Console** view.	Deselect **Append to Natural console**.
Change the log pattern to format the logging information.	Type the pattern in **Pattern**.

4 Select **OK** to save the preferences.

Set Natural Preferences

This section describes how to set preferences for Natural.

To set Natural preferences

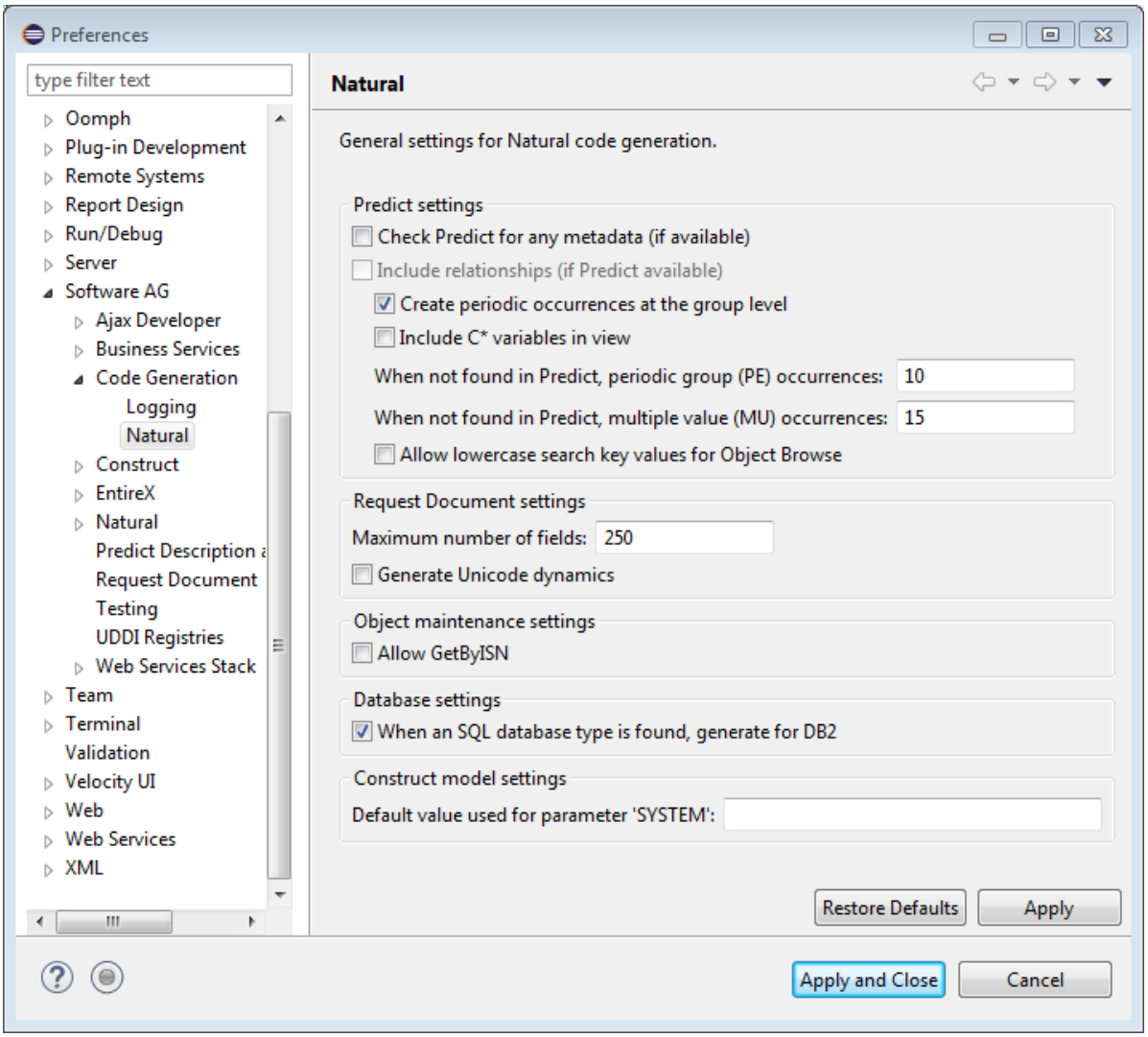
1 Select **Preferences** on the **Window** menu.

The **Preferences** window is displayed.

2 Expand the **Software AG** root node.

3 Select **Code Generation > Natural**.

The **Natural** preferences are displayed. For example:



Using this window, you can:

Task	Procedure
Check Predict for additional metadata.	Select **Check Predict for additional metadata**. **Note:** If this option is selected, three options in the **Predict Settings** section are disabled. If this option is not selected, use these options to simulate the Predict data.
Include relationship data from Predict (if available on the server).	Select **Include relationships**. **Note:** 1. To enable this option, select **Check Predict for additional metadata**. 2. Currently, relationships for DB2 objects are not processed. 3. The defined field names in the related Predict files have to be unique. Otherwise the generated LDAs and PDAs cannot be stowed successfully. For further information please refer to *Qualifying Data Structures* in the *Programming Guide* provided under *Natural Language for Windows*.

Suppress the creation of periodic occurrences at the group level.	Deselect **Create periodic occurrences at the group level**.
	
	For example, a DDM containing a periodic group (PE) named INCOME with four occurrences can be represented as follows:
	
	■ At a group level
	
	For example:
	
	`2 INCOME (1:4)`
	`  3 SALARY (P5)`
	`  3 CURRENCY (A3)`
	
	■ Not at a group level
	
	For example:
	
	`2 INCOME`
	`  3 SALARY(P5/4)`
	`  3 CURRENCY (A3/4)`

Include all C* variables in the view.	Select **Include C* variables in view**.
	
	If this option is selected, the C* variables are generated into the code to determine the number of occurrences of a periodic group. For example:
	
	`2 C*INCOME`
	`2 INCOME`

Task	Procedure
	`3 SALARY(P5/4)` `3 CURRENCY (A3/4)`
Change the maximum number of occurrences for a periodic group when not found in Predict.	Type the number in **When not found in Predict, periodic group (PE) occurrences**. The number of occurrences of a periodic group is not stored with the DDM and the maximum number of occurrences could be too large to use. To solve this problem, you can define the maximum number of PE occurrences in this field. **Note:** If **Check Predict for additional metadata** is not selected, or if 0 is returned from Predict, the value in this field will be used.
Change the maximum number of occurrences for a multiple-valued field when not found in Predict.	Type the number in **When not found in Predict, multiple value (MU) occurrences**. The number of occurrences of a multiple-valued field is not stored with the DDM and the maximum number of occurrences may be too large to use. To solve this problem, you can define the maximum number of MU occurrences in this field. **Note:** If **Check Predict for additional metadata** is not selected, or if 0 is returned from Predict, the value in this field will be used.
Allow search keys to be entered in lower case for an object-browse subprogram.	Select **Allow lowercase search key values for Object Browse**. By default, the object browse subprogram will convert the starting values for all supplied alphanumeric key components to upper case. If this option is selected, the ALLOW-LOWER-CASE option is generated for all keys and the input values can include lower case characters. For example, if the database contains both upper case and lower case values for the BUSINESS-NAME field (for example, iXpress and IBM) and you select this option, either lower case or upper case input values can be used in a search (for example, "I*" for iXpress and "I*" for IBM).
Change the maximum number of fields generated for a REQUEST-DOCUMENT subprogram.	Type a new number in **Maximum number of fields**. A REQUEST-DOCUMENT subprogram can generate a large amount of code, which may cause memory errors. To avoid this, you can use this option to place restrictions on the REQUEST-DOCUMENT Client code generator.
Generate the dynamics to support Unicode fields.	Select **Generate Unicode dynamics**. This option allows the REQUEST-DOCUMENT subprogram to send and receive Unicode data. **Note:** The Natural server must be configured for Unicode.

Task	Procedure
Allow the GetByISN option for an Object Maint subprogram.	Select **Allow GetByISN**. This option is available for Adabas files. If this option is selected, data can be retrieved using the ISN. Although extra code is generated, performance speed will be enhanced.
Suppress the generation of DB2 code for SQL database types.	Deselect **When an SQL database type is found, generate for DB2**.
Construct model settings	Can be used to change the default name to be used for standard parameter `SYSTEM`. Enter a different default name to be used for parameter `SYSTEM`. **Note:** The following description refers to the generation of a new Construct model on the NaturalONE client based on Construct installed on the Natural server as described in section *Access the Client Generation Wizards*. By default, the library name which is specified in the client generation wizards, e.g. as described in section *Generate the Modules* is used for the standard parameter `SYSTEM`. The system name must be alphanumeric and not exceed 32 characters in length. The combination of the module name and system name is used as a key to access help information for the generated module.

4 Select **OK** to save the preferences.

9 Customize the Code Generators

- Export the Supplied Templates 64
- Customize a Supplied Template 67

The code generators supplied with NaturalONE use Velocity templates, which are embedded in the .jar file created during the build process. To customize the templates, you can copy the embedded templates from the .jar file to your **custom-templates** folder and modify the template there. Velocity will check this folder first for the template. If it exists, it will be used by the code generator.

This section describes how to customize the code generators.

Export the Supplied Templates

To export the supplied templates

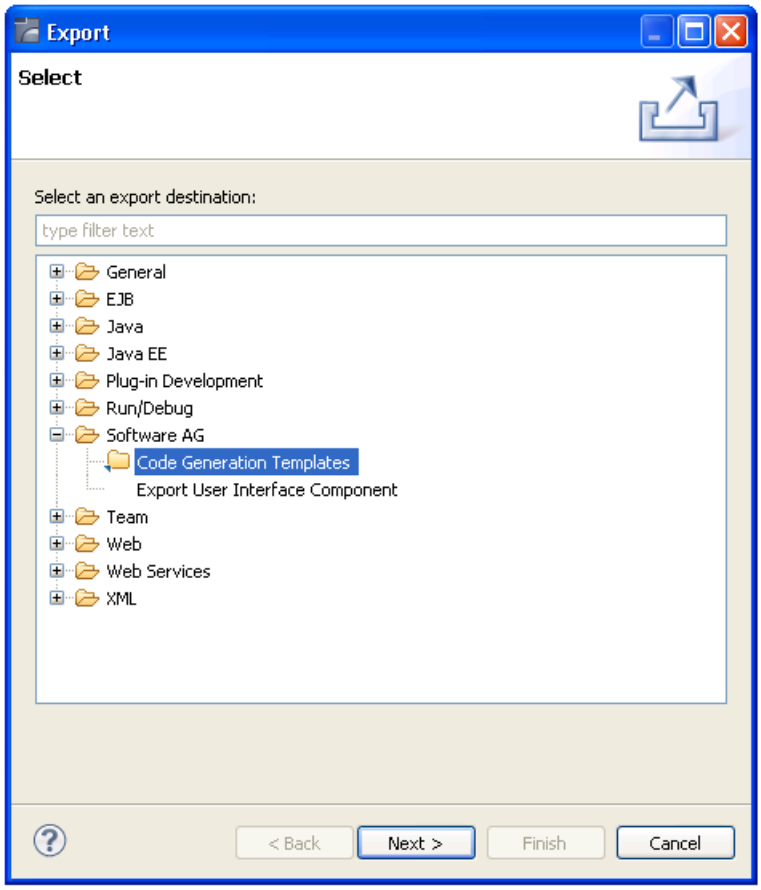
1 Select **Export** on the **File** menu.

The **Export** window is displayed.

2 Expand the **Software AG** root node.

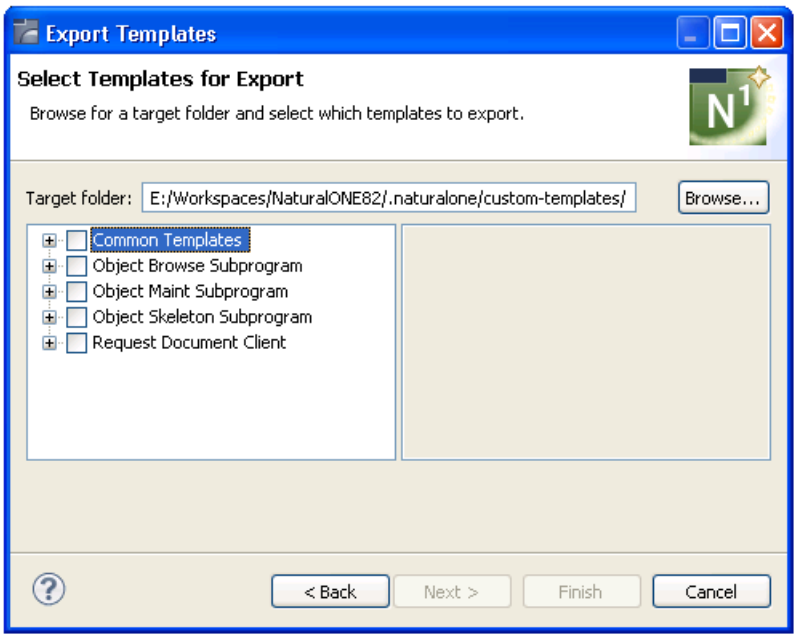
3 Select **Code Generation Templates**.

For example:



4 Select **Next**.

The **Select Templates for Export** panel is displayed. For example:



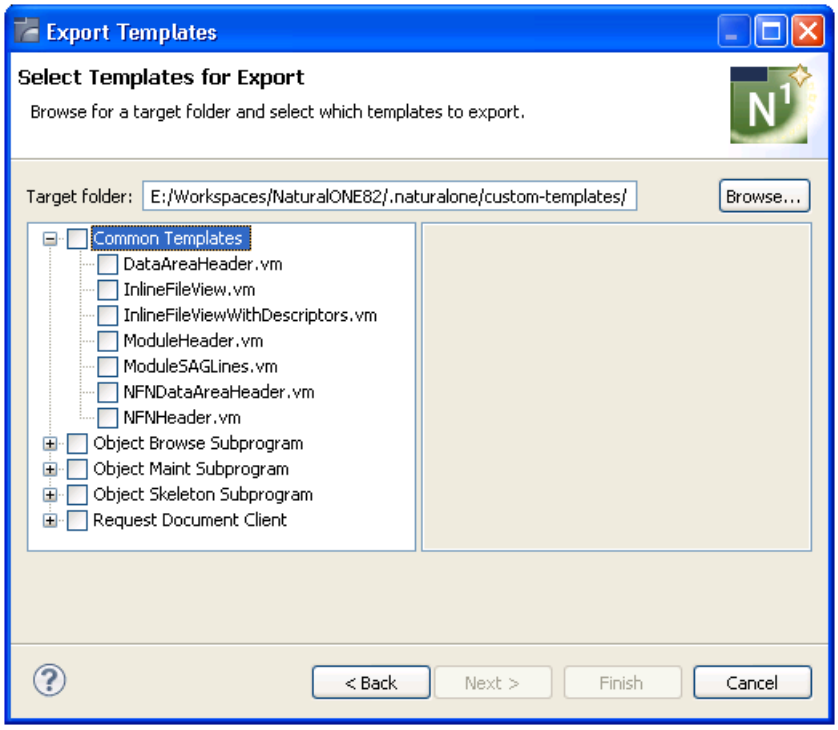
This panel displays the default target folder for the templates, as well as the templates available for export.

Note: You can change the default target folder in the **Preferences** window for Code Generation. For information, see *Set Code Generation Preferences*.

5 Select the templates you want to export.

- To select all templates in a template node, select the node (for example, if you select the **Common Templates** root node, all templates within that node will be selected).
- To select individual templates, expand a template root node and select the template.

  For example:



6 Select **Finish**.

The templates are exported to the selected target folder.

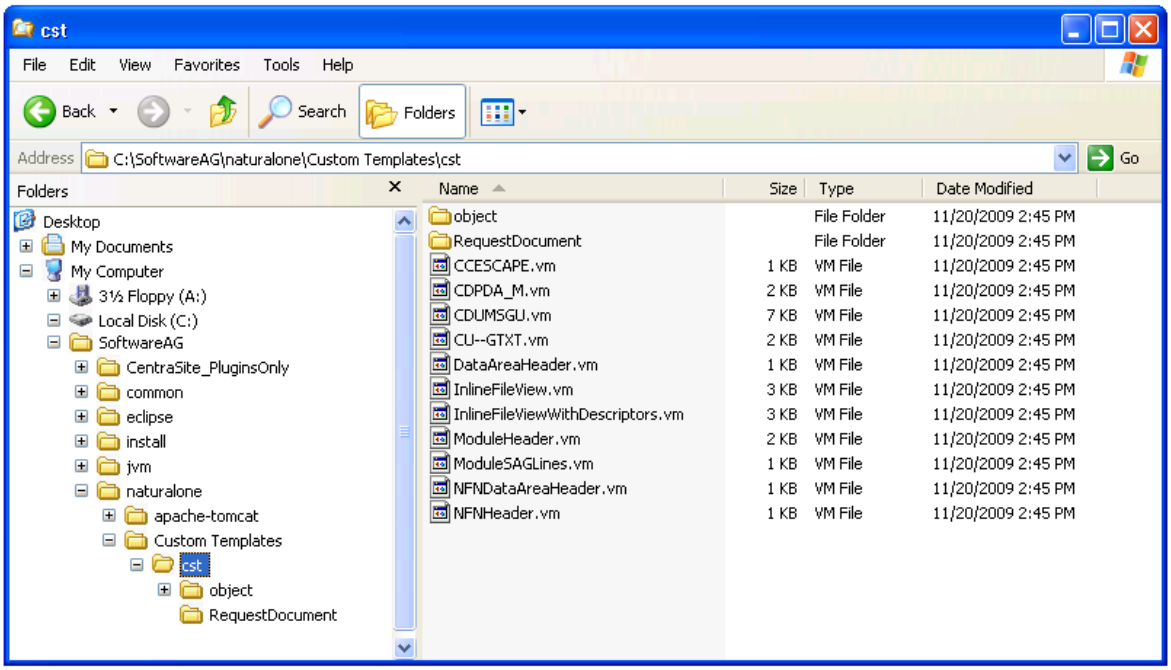
Note: You cannot change the functionality of the internal Java code, you can only modify the templates.

Customize a Supplied Template

To customize a template

1 Select the template in the **custom-templates > cst** folder.

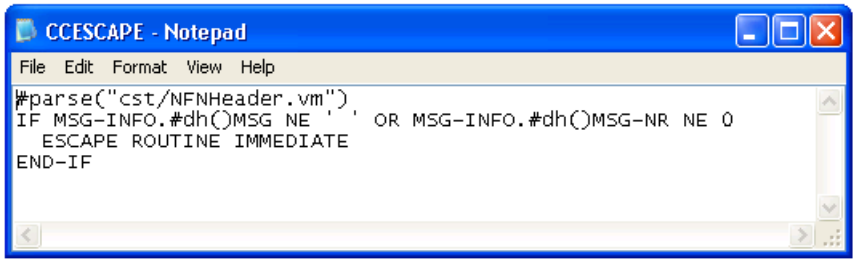
For example:



Note: Custom templates are stored in the folder specified in the **Preferences** window for Code Generation. For information, see *Set Code Generation Preferences*.

2 Open the template you want to modify.

For example:



3 Modify the template.

4 Save the changes.

II Using Natural Construct

This part describes the Natural Construct component supplied with NaturalONE.

The following topics are covered:

Introduction
Natural Construct Generation
Natural Construct Administration
Set Natural Construct Preferences

10 Introduction

- Supplied Client Generation Wizards 72
- Requirements 74
- Perform Standard Actions on Natural Construct Resources 76
- Use the Dependencies View 81

The Natural Construct component for NaturalONE provides access from Eclipse to Natural Construct on the server. This access includes the modeling functionality in the SYSCST library, as well as Eclipse wizards corresponding to a subset of the most common Natural Construct models in your server installation. The Eclipse wizards collect the model specifications and pass this information to the Natural server to generate the code, which is then returned to the local project that was selected as the target on a wizard panel.

Note: You must have Natural Construct installed in a server environment to use this component.

This type of code generation is different from the local, non-server based generation implemented using Velocity templates (for example, REQUEST-DOCUMENT Client, Object Maint, Object Browse, and Object Skeleton; see *Using the Code Generation Component*). The Natural Construct component allows you to use Natural Construct models on the server in NaturalONE, as well as create Eclipse wizards for them (including customized ones).

Note: To install the Natural Construct component for NaturalONE, **Designer > NaturalONE > Natural Construct** must be selected in the installation tree for the installer. **NaturalONE > Natural Construct** is selected by default when you select **Designer** in the installation tree.

Supplied Client Generation Wizards

The Natural Construct component for NaturalONE supplies client generation wizards for the following Natural Construct models on the server:

Model	Generates	For Information
BATCH	Batch programs can be generated for large volumes of output, routed to a printer or a terminal.	*Batch Wizard*
BROWSE	Browse program that reads a file in logical order and displays record values on the screen.	*Browse/Browse-Select Wizards*
BROWSE-SELECT	Browse-select program that reads a file in logical order, displays record values on the screen, and allows the user to specify which set of commands are executed.	*Browse/Browse-Select Wizards*
BROWSE-SELECT-HELPR	Browse-select helproutine that enables the user to select a field value from a list of valid values.	*Browse/Browse-Select Wizards*
BROWSE-SELECT-SUBP	Browse-select subprogram that is invoked as a sub-function of another program. For example, you can use	*Browse/Browse-Select Wizards*

Model	Generates	For Information
	a browse-select subprogram to perform the Browse action for a maintenance program, in which case, the maintenance program invokes the subprogram without disturbing the current state of the panel.	
BROWSE-SUBP	Browse subprogram that is invoked as a sub-function of another program.	*Browse/Browse-Select Wizards*
DRIVER	Driver program that executes a helproutine or subprogram for testing purposes.	*Driver Wizard*
MAINT	Maintenance program that maintains a file using a unique key and, optionally, a related secondary file. The Maint wizard generates the code necessary to scroll through the MU/PE fields of a primary file or the records of a secondary file.	*Maint Wizard*
MENU	Menu program that presents users with several choices in the form of a menu. The user enters a code for one of the choices to invoke a predefined function.	*Menu Wizard*
OBJECT-BROWSE-DIALOG	Object-browse dialog component of an object-maintenance process that works with the object-browse subprogram to provide the browse functionality for a Natural object.	*Object-Browse-Dialog Wizard*
OBJECT-BROWSE-SELECT-SUBP	Object-browse-select subprogram and corresponding parameter data areas that provide the browse functionality for a Natural object. This model is similar to the OBJECT-BROWSE-SUBP model, except the generated object-browse-select subprogram can accommodate a client/server environment and a subprogram proxy can be used to access the generated code as a business service.	*Object-Browse-Select-Subp Wizard*
OBJECT-BROWSE-SUBP	Object-browse subprogram and corresponding parameter data areas	*Object-Browse-Subp Wizard*

Model	Generates	For Information
	that provide the browse functionality for a Natural object.	
OBJECT-MAINT-DIALOG	Object-maintenance dialog component of an object-maintenance process. The dialog component (Natural program) communicates with the user and invokes methods (data actions) implemented by the object-maintenance subprogram.	*Object-Maint-Dialog Wizard*
OBJECT-MAINT-ENHANCED-SUBP	Object-maintenance subprogram and corresponding parameter data areas that update all entities within a Natural object. Similar to the Object-Maint-Subp wizard, the main difference between these wizards is that the Object-Maint-Enhanced-Subp wizard will generate large fields in the object PDA as dynamic fields.	*Object-Maint-Enhanced-Subp Wizard*
OBJECT-MAINT-SUBP	Object-maintenance subprogram and corresponding parameter data areas that update all entities within a Natural object.	*Object-Maint-Subp Wizard*
QUIT	Quit program that releases resources used by an application. It displays a confirmation window that overlays the host panel and gives users the option of quitting an application entirely or resuming where they left off.	*Quit Wizard*
STARTUP	Startup program (often named Menu) that initializes global variables and invokes the main menu program.	*Startup Wizard*

Requirements

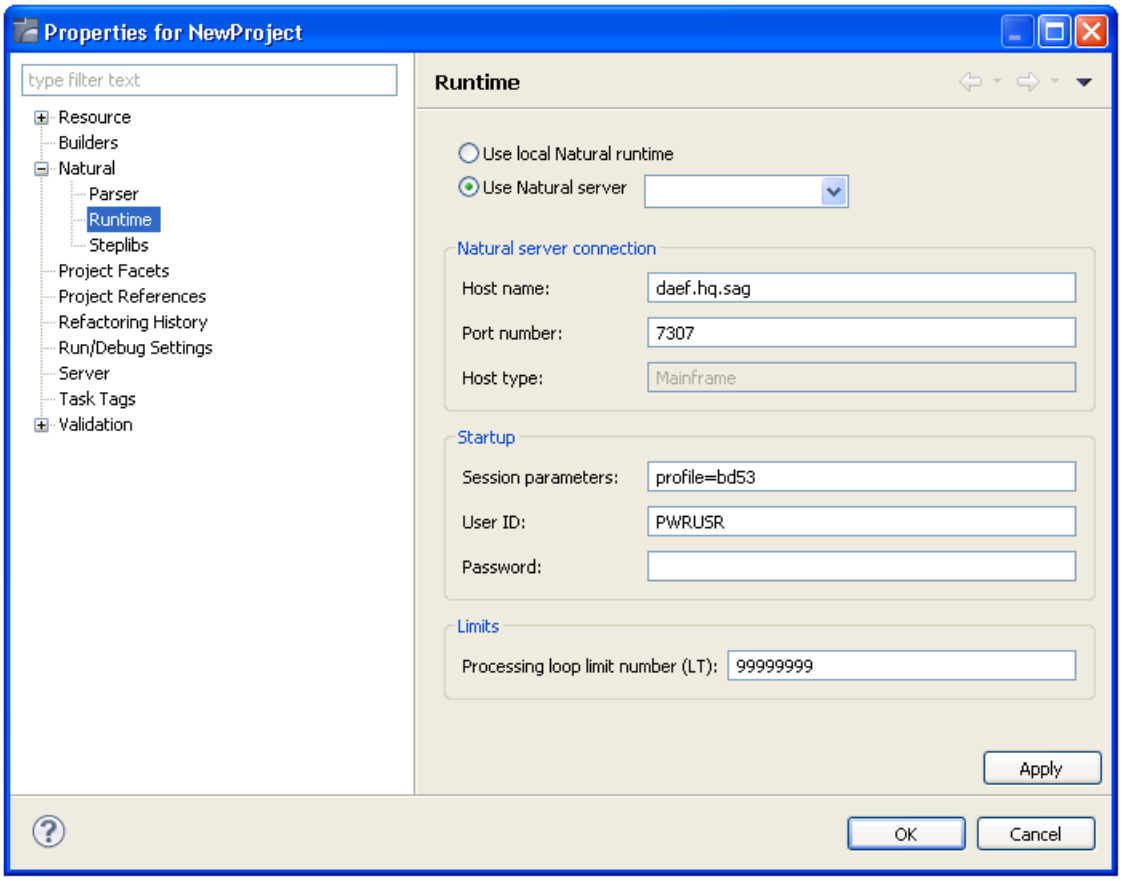
To use the Natural Construct code generation features in NaturalONE, the following requirements must be met:

- Your NaturalONE environment must be mapped to a server in the **Natural Server** view that contains a version of Natural Construct 5.3, service pack 8 or higher.

- Projects in your workspace must be connected to the server containing Natural Construct; projects mapped to the local Natural runtime environment cannot be used to generate Natural Construct modules.

  The target Natural project must be configured to a remote environment. If you select a Natural project that is mapped to a local environment, an error will be displayed. When you change the target project to a valid remote project, the clear subprogram will be called using the connection settings for the valid project.

- The **Natural >Runtime** setting in the **Properties** window for the project must point to the **Natural Server** connection containing the Natural Construct installation. For example:



Perform Standard Actions on Natural Construct Resources

You can use the **Natural Server** view to copy/paste, delete, or move Natural Construct resources on the server. The action will be performed in the mapped environment for the selected node(s).

This section covers the following topics:

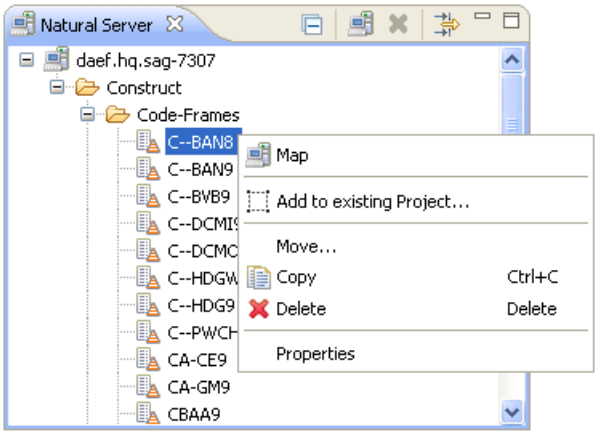
- Perform Actions on Code Frames
- Perform Actions on Models

Perform Actions on Code Frames

To perform actions on one or more code frames

1 Open the context menu for the code frame(s) in the **Natural Server** view.

For example:



2 Select one of the actions listed.

The available actions are:

Action	Description
Move	Removes the selected code frame(s) from the current mapped environment and adds it to a target mapped environment. For information, see *Move a Code Frame*.
Copy	Copies the selected code frame(s) to the clipboard in anticipation of a Paste action. For information, see *Copy a Code Frame*.
Delete	Removes the selected code frame(s) from the current mapped environment. For information, see *Delete a Code Frame*.

Move a Code Frame

This section describes how to move one or more code frame(s) from the current mapped environment to a target mapped environment.

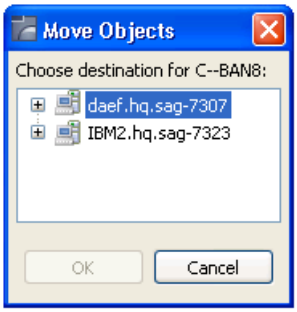
Note: A code frame cannot be moved within the same mapped environment.

To move one or more code frames

1 Open the context menu for the code frame(s) in the **Natural Server** view.

2 Select **Move**.

The **Move Objects** window is displayed. For example:



This window lists the connection nodes for the available mapped environments.

3 Expand the connection node for the environment into which you want to move the code frame(s).

4 Select the **Construct** or **Code-Frames** root node.

5 Select **OK**.

A progress window is displayed while the code frames are removed from the current mapped environment and copied to the target mapped environment.

Copy a Code Frame

This section describes how to copy one or more code frames to the clipboard and then paste the frame(s) into a target mapped environment.

To copy one or more code frames

1 Open the context menu for the code frame(s) in the **Natural Server** view.

2 Select **Copy**.

3 Open the context menu for the **Construct** or **Code-Frames** root node into which you want to copy the code frame(s).

4 Select **Paste**.

The frames are copied to the target mapped environment.

Delete a Code Frame

This section describes how to remove one or more code frames from the current mapped environment.

To delete one or more code frames

1 Open the context menu for the code frame(s) in the **Natural Server** view.

2 Select **Delete**.

A confirmation window is displayed to confirm the action.

3 Select **Yes**.

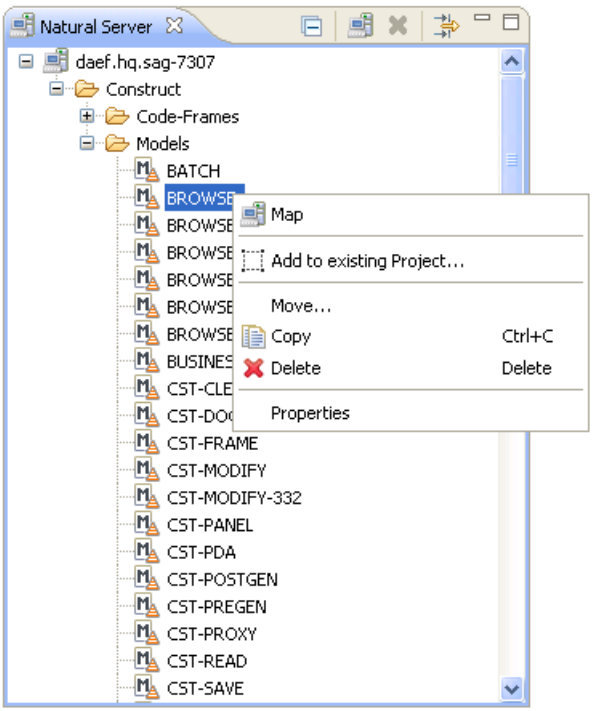
The frames are removed from the current mapped environment.

Perform Actions on Models

To perform actions on one or more Natural Construct models

1 Open the context menu for the model(s) in the **Natural Server** view.

For example:



2 Select one of the actions listed.

The available actions are:

Action	Description
Move	Removes the selected Construct model from the current mapped environment and adds it to a target mapped environment. For information, see *Move a Construct Model*.
Copy	Copies the selected model(s) to the clipboard in anticipation of a Paste action. For information, see *Copy a Construct Model*.
Delete	Removes the selected model(s) from the current mapped environment. For information, see *Delete a Construct Model*.

Move a Construct Model

This section describes how to move one or more model(s) from the **Models** root node in the current mapped environment to a target mapped environment.

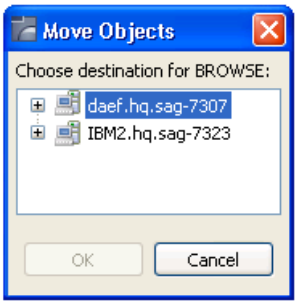
Note: A Construct model cannot be moved within the same mapped environment.

To move one or more Construct models

1 Open the context menu for the model(s) in the **Natural Server** view.

2 Select **Move**.

The **Move Objects** window is displayed. For example:



This window lists the connection nodes for the available mapped environments.

3 Expand the connection node for the environment into which you want to move the model(s).

4 Select the **Construct** or **Models** root node into which you want to move the model(s).

5 Select **OK**.

A progress window is displayed while the models are removed from the current mapped environment and copied to the target mapped environment.

Copy a Construct Model

This section describes how to copy one or more models to the clipboard and then paste the model(s) into a target mapped environment.

To copy one or more Construct models

1 Open the context menu for the model(s) in the **Natural Server** view.

2 Select **Copy**.

3 Open the context menu for the **Construct** or **Models** root node into which you want to copy the model(s).

4 Select **Paste**.

The models are copied to the target mapped environment.

Delete a Construct Model

This section describes how to remove one or more models from the current domain.

To delete one or more Construct models

1 Open the context menu for the model(s) in the **Natural Server** view.

2 Select **Delete**.

A confirmation window is displayed to confirm the action.

3 Select **Yes**.

The models are removed from the current mapped environment.

Use the Dependencies View

When a Construct resource (for example, a Construct model, code frame, etc.) is open in the editor, the **Dependencies** view displays dependencies between that resource and other Construct resources and/or Natural resources. This section describes the child nodes contributed to the view by the Construct-related resources.

The following topics are covered:

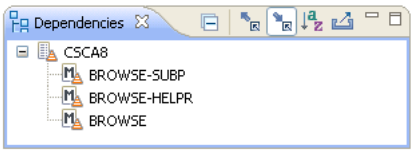
- Construct Resources
- Related Natural Resources

Notes:

1. Select to sort the resources alphabetically.
2. Select to export a textual representation of the visible nodes in the view to a file.
3. When a supporting resource cannot be found locally using the project steplib chain and project references, "<Unknown>" is displayed with the name of the resource (see above). If the unknown module(s) is not shipped with the Construct runtime project, either manually download it from the server or create it locally. If the module(s) is shipped with the Construct runtime project, add the project. For information, see *Add the Construct Runtime Project*.
4. For more information about the **Dependencies** view, see the description of the source editor in *Using NaturalONE*.

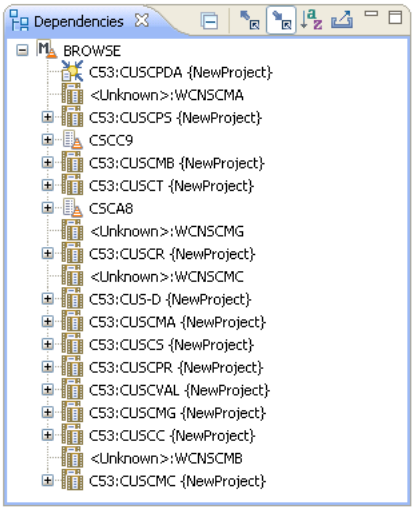
Construct Resources

When a Construct resource is open in the editor, the root node displays the name of the resource. In caller mode (), child nodes are contributed to the **Dependencies** view for each resource that depends on that Construct resource. For example, when a code frame is open in the editor, the child nodes display any Construct models or other code frames that depend on that code frame:



In this example, three Construct models (BROWSE-SUBP, BROWSE-HELPR, and BROWSE) depend on the CSCA8 code frame.

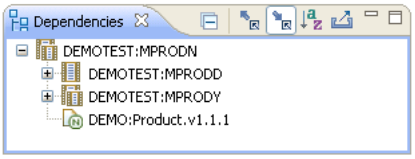
In callee mode (), Construct and Natural nodes are contributed to the view for each resource the Construct resource depends on. For example, when a Construct model is open in the editor, the child nodes will display any code frames, PDAs, subprograms and Construct models that this model depends on. For example:



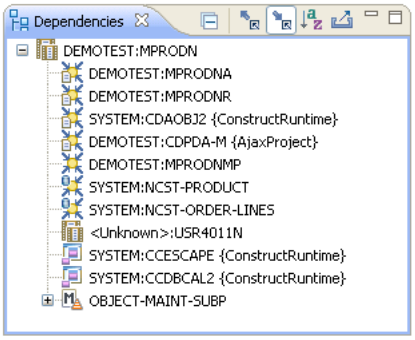
In this example, the Construct model named BROWSE depends on two code frames (CSCC9 and CSCA8), a PDA named CUSCPDA in the C53 library and many subprograms.

Related Natural Resources

When a Natural subprogram is open in the editor, the root node displays the name of the subprogram, as well as the name of the library in which it is located. In caller mode (), child nodes are contributed to the **Dependencies** view for each Construct-related resource that depends on this subprogram (such as a Construct model, code frames, etc.). For example:



In callee mode (), a Construct model node is contributed to the view if the subprogram was generated by a Construct model. For example:



11 Natural Construct Generation

- Access the Client Generation Wizards 86
- Generate the Modules 87
- Common Wizard Specifications and Development Tasks 91
- Example of Generating a Program 201
- Regenerate Natural Construct-Generated Modules 204

This section describes how to use the Natural Construct client generation wizards to generate Natural modules, as well as how to define user exits for additional processing that is preserved during regeneration.

Access the Client Generation Wizards

Notes:

1. When you have defined customized modules on the server side, downloaded them and generated a new Construct model UI on the client side, the customized models will also appear in the list of the available Construct models. For details see *Create and Maintain a Natural Construct Model UI*.
2. When using the function **New Using Construct Model** for a customized model, the models must have defined a Stream and Validate subprogram. Otherwise the module generation on the server side will fail.
3. The following description refers to the generation of a new Construct model on the NaturalONE client based on Construct installed on the Natural server.

To access the client generation wizards

1 Open the context menu in the **Project Explorer** view for the NaturalONE project in which you want to generate the modules.

Or:

Open the context menu in the **Project Explorer** view for the library in which you want to generate the modules.

Note: You can also access the wizards using standard NaturalONE functionality (i.e., through the **File** menu or using the **New** toolbar option.

2 Select **Code Generation > New Using Construct Model**.

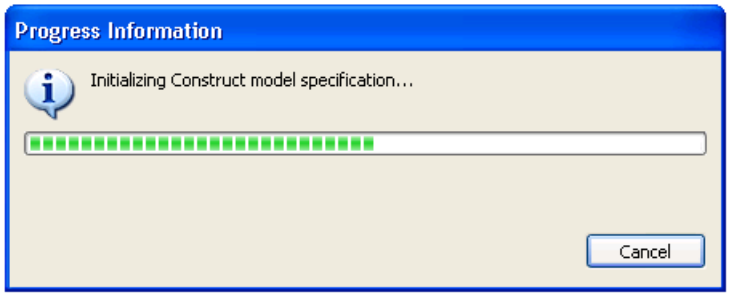
A list of the supplied client generation wizards is displayed.

Note: The list of wizards displayed in this menu is based on which Construct version is installed on the target server. For example, Construct V8.2 wizards will not be available if the project is attached to a Construct V5.3 server.

3 Select the wizard you want to use.

The **Progress Information** window is displayed, indicating progress as the model specification PDA is initialized by the model's clear subprogram on the server to set the model defaults

(the same process initiated when you enter "NCSTG" from a character-based Natural connection). For example:

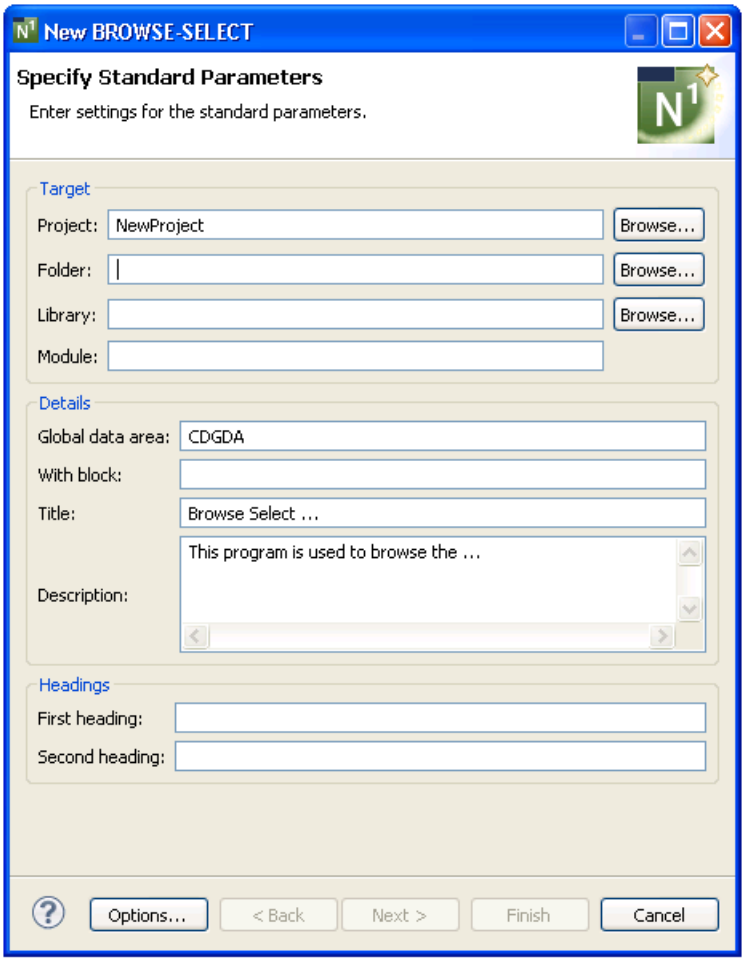


The clear subprogram will only be called when all the following conditions are true:

- The selected target project is valid and connected to a remote Natural environment.
- The generation mode is set to "New" (not regeneration).
- The clear subprogram has never been called or the last successful call to the clear subprogram targeted a different project.

Generate the Modules

When initialization is complete, the first specification panel for the selected client generation wizard is displayed. For example:



The module parameters are grouped by topic and **Browse** buttons are available when selecting existing resources. After specifying the first parameter on a panel, messages are displayed indicating the next required parameter. The **Next** button will only be enabled when all required parameters have been specified on the current panel have been specified; the **Finish** button will only be enabled when all required parameters have been specified for the current wizard.

To generate a module

1 Specify all required parameters and any optional parameters on the first panel for the selected wizard.

2 Select **Next**.

Or:

Select **Finish**.

The generation process begins. By default, progress is detailed in messages displayed near the bottom of the panel. Once generation is complete, the code is downloaded to the client.

The generated source is displayed in the editor and the available user exits are displayed in the **Outline** view.

3 Save the generated module(s).

4 Use standard NaturalONE functionality to upload the generated module(s) to the server.

The following table lists the supplied Natural Construct client generation wizards and where you can find information on the specification parameters for each wizard:

Wizard	Information
BATCH	*Batch Wizard*
BROWSE	*Browse/Browse-Select Wizards*
BROWSE-SELECT	*Browse/Browse-Select Wizards*
BROWSE-SELECT-HELPR	*Browse/Browse-Select Wizards*
BROWSE-SELECT-SUBP	*Browse/Browse-Select Wizards*
BROWSE-SUBP	*Browse/Browse-Select Wizards*
DRIVER	*Driver Wizard*
MAINT	*Maint Wizard*
MENU	*Menu Wizard*
OBJECT-BROWSE-DIALOG	*Object-Browse-Dialog Wizard*
OBJECT-BROWSE-SELECT-SUBP	*Object-Browse-Select-Subp Wizard*
OBJECT-BROWSE-SUBP	*Object-Browse-Subp Wizard*
OBJECT-MAINT-DIALOG	*Object-Maint-Dialog Wizard*
OBJECT-MAINT-ENHANCED-SUBP	*Object-Maint-Enhanced-Subp Wizard*
OBJECT-MAINT-SUBP	*Object-Maint-Subp Wizard*
QUIT	*Quit Wizard*
STARTUP	*Startup Wizard*

Notes:

1. During generation, the wizard determines whether the Construct runtime project is available locally, and if it is not, prompts you to add it. For information, see *Add the Construct Runtime Project*.
2. To change the default generation options and/or set other generation options, see *Generation Options*.
3. If the generated module is a subprogram, you can test it using the NaturalONE Testing option. For information, see *Test a Subprogram Directly* in *Application Testing*.
4. For information about adding custom code within user exits, see *Defining User Exits*.

Generation Options

In addition to the standard navigation buttons available at the bottom of the wizard panels, an **Options** button is available to define generation options.

To define generation options

1 Select **Options** on the wizard specification panel.

The **Generation Options** window is displayed. For example:

Using this window, you can:

Task	Procedure
Disable the display of generation status messages.	Deselect **Display generation status messages**. The generation status messages indicate which module is being invoked at each stage of the generation process.
Display the generation status messages as text (for example, "starting" and "ending").	Select **Display generation status messages as text**. By default, the messages are displayed with arrows "-->" (starting) and "<--" (ending).
Write embedded statements to the source buffer as part of the generated module.	Select **Include embedded statements in generated code**. Embedded statements indicate where the lines of code being written originated and the name of the code frame, generation subprogram, or sample subprogram that produced the code.

2 Select **OK** to save the generation options.

Common Wizard Specifications and Development Tasks

The specification parameters listed on the wizard panels correspond with those on panels for the Natural Construct models on the server. This section describes the common specifications for the Natural Construct wizards and how to perform common development tasks.

Notes:

1. For an example of using a client generation wizard to generate a module, see ***Example of Generating a Program***.
2. For information about specific parameters for the wizards, see the applicable model in the *Natural Construct Generation* guide.

This section covers the following topics:

- Batch Wizard
- Browse/Browse-Select Wizards
- Driver Wizard
- Maint Wizard
- Menu Wizard
- Object-Browse-Dialog Wizard
- Object-Browse-Select-Subp Wizard
- Object-Browse-Subp Wizard
- Object-Maint-Dialog Wizard
- Object-Maint-Enhanced-Subp Wizard
- Object-Maint-Subp Wizard
- Quit Wizard
- Startup Wizard
- Change the Dynamic Attribute Characters
- Change the Window Settings
- Select a Message Number
- Specify Common Parameters
- Specify International Parameters
- Specify Screen Parameters

- Specify Standard Parameters

Batch Wizard

This section describes the specification parameters for the Batch wizard. With the Batch Wizard, batch programs can be generated. Commonly, these can produce large volumes of output, which the programs route to a printer or a terminal.

A batch program uses a primary key value to read the contents of a primary file in logical order. The primary key value can be a descriptor, superdescriptor, or subdescriptor, and does not have to be unique. These programs differ from online programs as they usually read or update large volumes of data and produce multiple reports. For multiple files, you can use the Batch model (which supports as many as two secondary files and two tertiary files). After selecting the files to use in your batch program, you can select additional, related files by indicating the Predict relationships to be process. File views returned from Predict can be used to create a report layout. Data read from the program can be written to four different reports, each of which may have a different format. To view the specifications for a sample batch program, refer to the NCOREP program in the Natural Construct demo system.

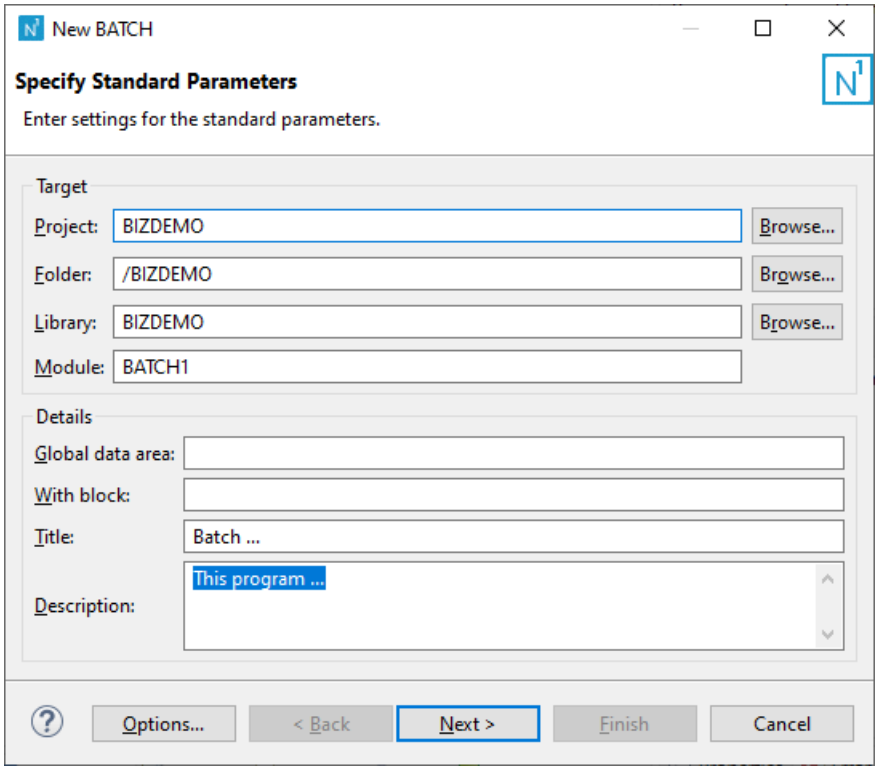
The Batch wizard provides the pages described below.

- Specify Standard Parameters
- Specify Report Heading Parameters
- Specify Basic Program Structure
- Specify Additional Parameters
- Specify Parameters for Primary file
- Specify Parameters for Secondary file (1)
- Specify Parameters for Secondary file (2)
- Specify Parameters for Tertiary file (1)
- Specify Parameters for Tertiary file (2)

Specify Standard Parameters

The **Specify Standard Parameters** panel is displayed when the wizard is invoked.

For information about these parameters, see ***Specify Standard Parameters***.



Specify Report Heading Parameters

This panel is divided into two sections:

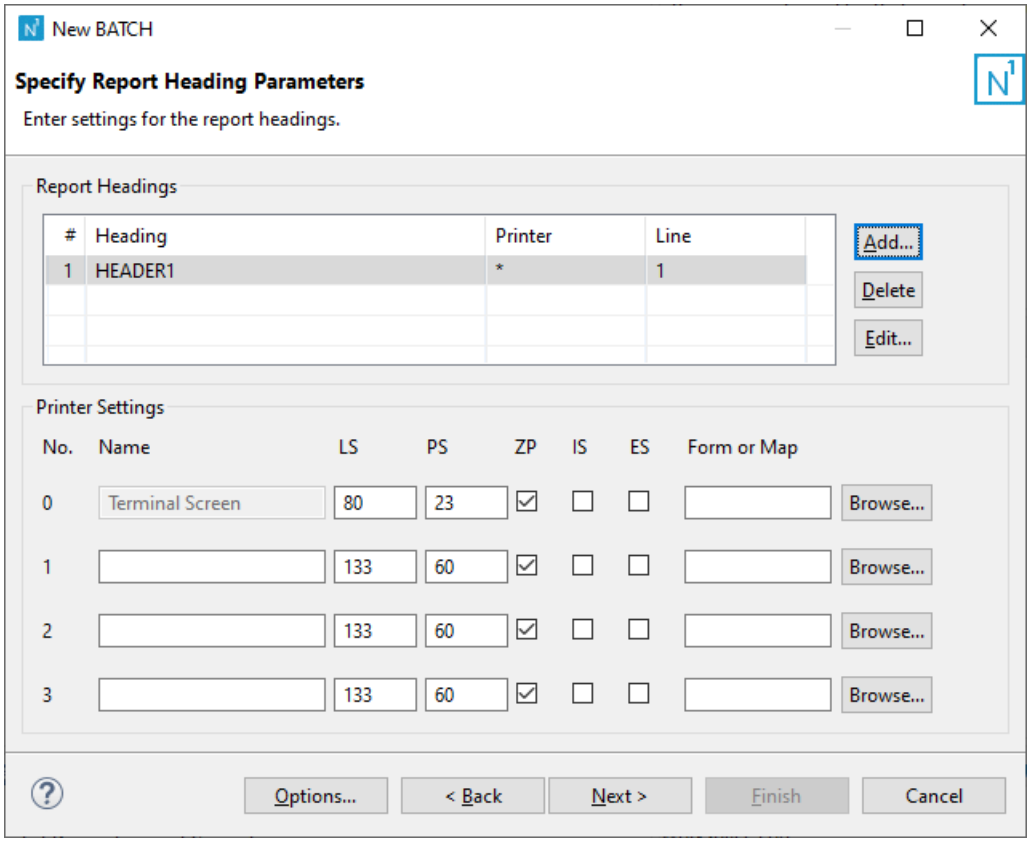
Use the upper section to enter report headings and indicate where they will appear on the report. Use the lower section to set up general formats for each report.

Panel reference

Field	Description
Report Headings	Headings for up to eight reports.
Printer	Printer file number for the corresponding report (see the Printer Name field description). If you type an asterisk (*) in the Printer field for a heading, the heading is displayed on all reports.
On Line	Line on which the heading is displayed. Type "1" for a first heading, "2" for a second heading, etc.
Printer Name	Where the output of the report is directed. The printer file numbers specify the following destinations: ■ (0) Output to terminal screen ■ (1) Output to first report ■ (2) Output to second report ■ (3) Output to third report

Field	Description
	Type the report name in the appropriate field. The name is used to set up a Natural DEFINE PRINTER statement. If this field is left blank, the name defaults to the printer file number (0, 1, 2, or 3).
LS	Line size. The number of characters that fit on one line on the report (default values are displayed).
PS	Page size. The number of lines per page (default values are displayed).
ZP	Zero printing. If this field is marked (by default), fields containing zeroes are printed on the report. If this field is not marked, zeroes are suppressed.
IS	Identical suppress. If this field is marked, identical entries on consecutive lines will not be printed on the report.
ES	Empty line suppression. If this field is marked, empty lines will not be printed on the report.
Form or Map	Name of the form or map layout used to build the report headings.

The following example shows the specification panel for the **Report Heading Parameters**:



Specify Basic Program Structure

Use this panel to specify the basic structure of the batch program. Four different program blocks are available:

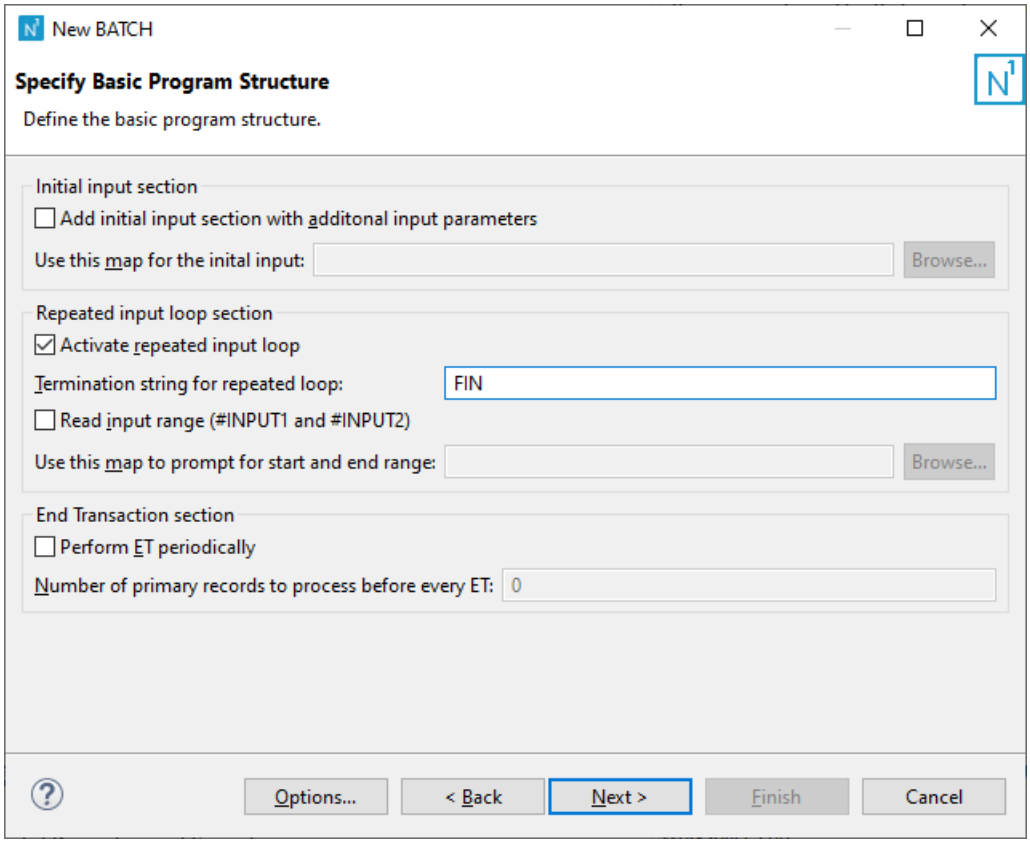
- INPUT statement for initial parameters
- REPEAT loop
- INPUT statement that prompts for a start and end range of primary key values within the REPEAT loop
- END TRANSACTION statement for updates

Panel reference

Field	Description
Initial input section	
Add initial input section with additional input parameters	If this field is marked, parameters are input at the beginning of the program (before any files are read). Specify the parameters on the Additional Parameters panel.
use this map for initial input	Name of the external map the program uses to input the parameters.
Repeated input loop section	
Activate repeated input loop	If this field is marked, a REPEAT loop reads the primary file for multiple ranges of records. By default, you can enter a single range of primary key values at the beginning of the program (after any additional input parameters). If you mark this field, primary key ranges will be continually input until the specified termination string is reached in the input stream.
Termination string for repeated loops	Termination string indicating when the program terminates. If you specified a map name in the INPUT USING MAP field (see above), you can use the #TERMINATED-STRING variable (modifiable, format A3) to input the termination string value.
Read input range	If this field is marked, the generated batch program reads only a portion of the file each time the REPEAT loop executes.
use this map for start and end range	Name of the external map the program uses to input the parameters. To specify a map name, use the #INPUT1.primary key variable to indicate the beginning of the range and the #INPUT2.primary key variable to indicate the end of the range (#INPUT1.ORDER-NUMBER, for example). **Note:** The primary key is the key of the primary file used in the batch program.
End transaction section	
Perform ET periodically	Mark this field to build a batch update program. An ET (END TRANSACTION) statement will be performed periodically to release records and update the database.

Field	Description
Number of primary records ... before every ET	If you marked the Perform ET field, specify the number of primary records to process before issuing an ET statement.

The following example shows the specification panel for the **Batch Program Structure**:



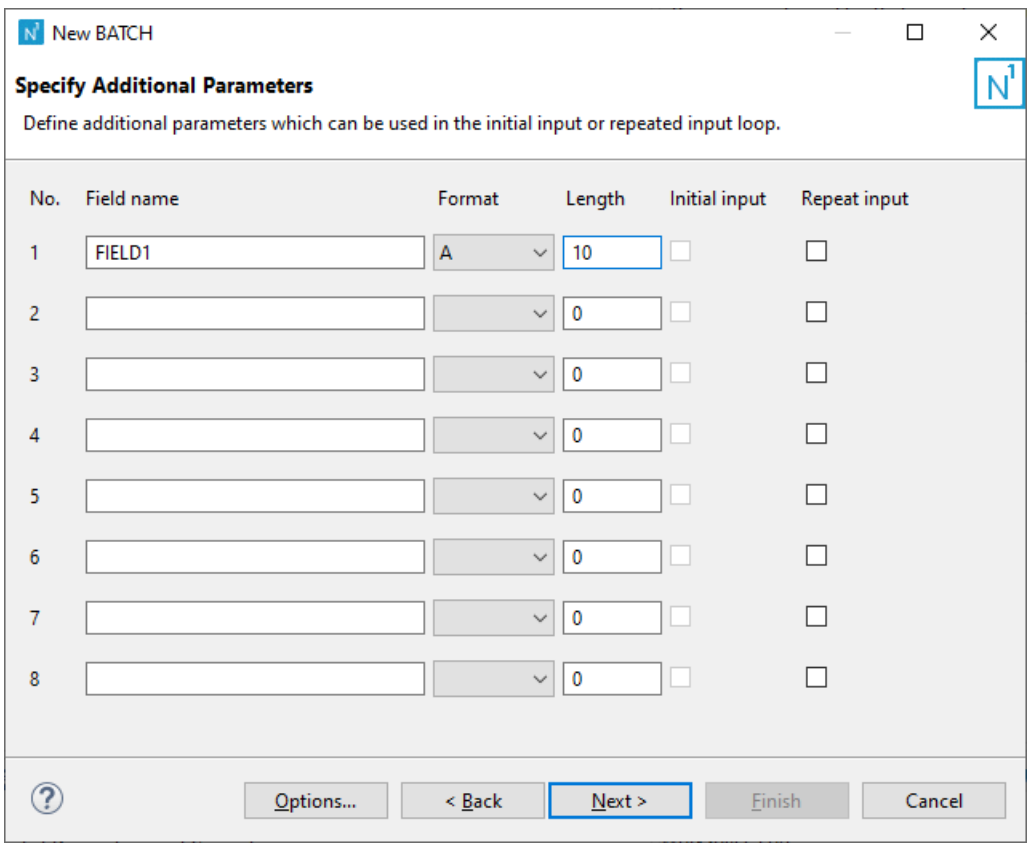
Specify Additional Parameters

Use this panel to specify the names of up to eight fields to input into the batch program, in addition to the range of key values for reading the primary file. These field names can be input either once at the beginning of the batch program or each time a new key range is specified. You can specify any field; the field does not have to be in the Predict file definition.

Panel reference

Field	Description
Field Name	Name(s) of the additional input field(s).
Format	Natural format and length. In the first portion of this field, type the code for the format of the corresponding field.
Format	Type the desired field length.
Initial INPUT	If the Additional inputs field on the Program Structure panel is marked, this field is available for modification. To apply the corresponding field name to the entire file, mark this field.
Repeat INPUT	If the Multiple inputs field on the Program Structure panel is marked, this field is available for modification. To apply the field name to the specified input range, mark this field.

The following example shows the specification panel for the Batch model, the **Additional Parameters** panel:



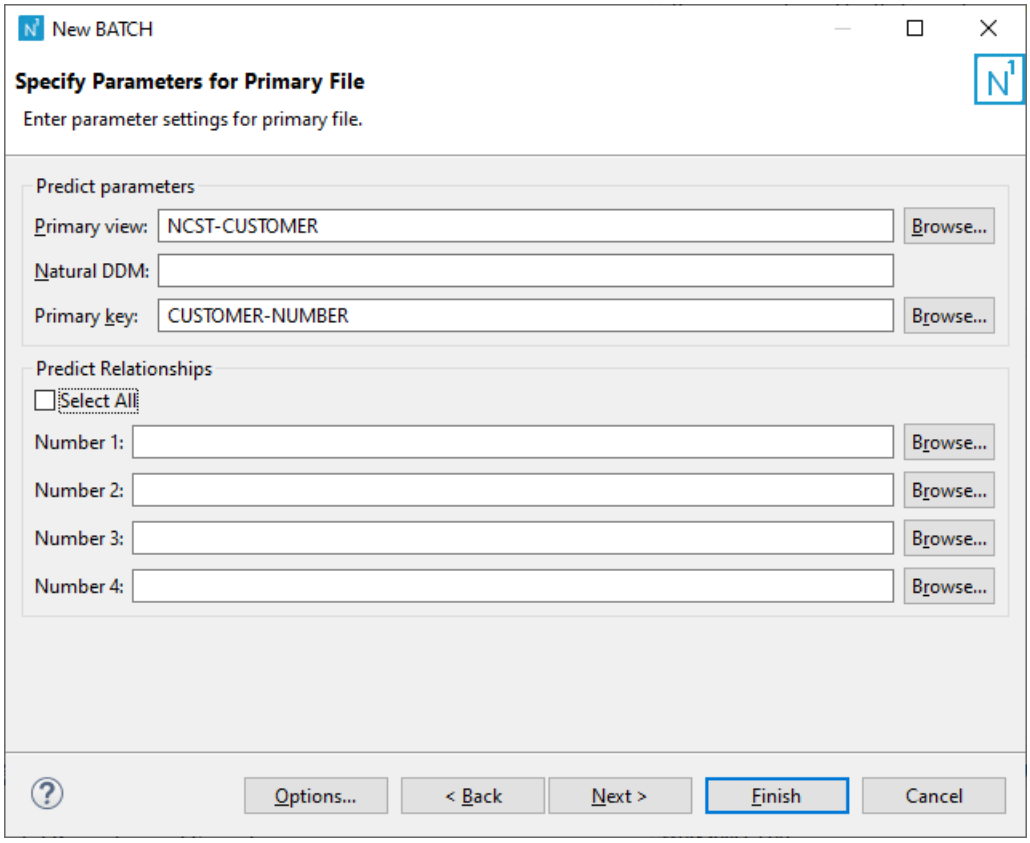
Specify Parameters for Primary file

Use this panel to specify the primary file and indicate which Predict relationships the batch program will use. If you want to define secondary and tertiary files you can navigate with the Next Button to the corresponding pages.

Panel reference

Field	Description
Predict parameters	
Primary view	View name driving the batch program. The specified view must be defined in Predict. The primary file type can be Adabas, DB2, VSAM, or sequential.
Natural (DDM)	Name of the data definition module (DDM) for the primary file. All fields in the primary file must be in the specified DDM. This field defaults to the name of the primary file, so you only need to specify a DDM if it is different from the primary file name.
Primary key	Key used to read the file in logical (rather than physical) sequence and to control scrolling. The key must be defined as a descriptor, superdescriptor, or subdescriptor in the Predict file definition. If the specified key does not exist in the corresponding Predict file, a message indicates that the Key field was not found in the file. **Note:** The primary key name is not applicable when the primary file is a sequential work file.
Predict Relationships	
Select all	If this field is marked, the generated program performs file lookups (joins) on all files related to the primary file in Predict, mark this field. In each relationship, the cardinality of the primary file must be N or CN, while the cardinality of the related file must be 1 or C. The update constraint type must be R (restricted update), and the delete constraint type can be blank or R. Only type N (Natural Construct) relationships are processed. In the generated batch program, the relationship name is used as the view name. Specify the related file name when selecting fields in the WRITE-FIELDS user exit. To avoid having to make manual changes after generating your user exit, change the related file name to the relationship name before generation. The generated fields will then have the correct prefix. **Note:** For DB2 users, type R (referential constraint) relationships are also processed.
Select specific	Names of up to four Predict relationships for the specified primary file.

The following example shows the **Primary File Parameters** panel:



Specify Parameters for Secondary file (1)

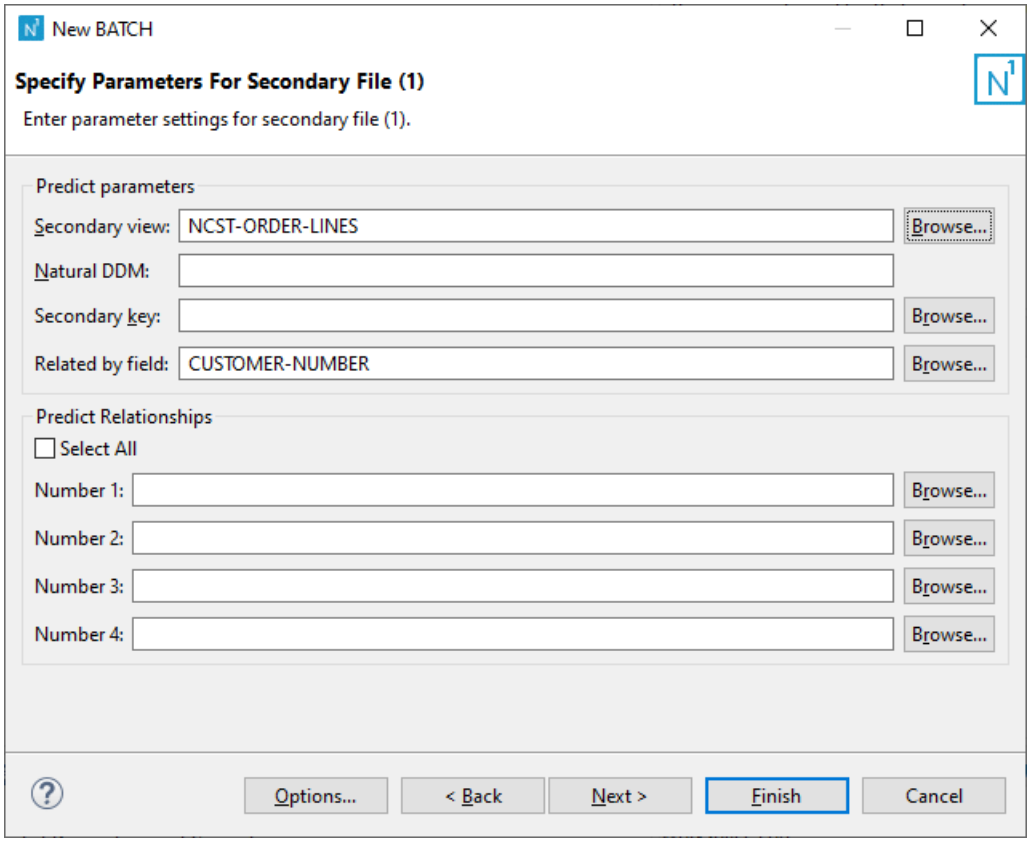
You can specify a secondary file, which will be logically coupled (joined) to the primary file in your batch program. The secondary file must be normally related to the primary file with cardinality 1:N.

Panel reference

Field	Description
Predict parameters	
Secondary view	Name of the secondary view, which must be defined in Predict. The secondary file type can be Adabas, DB2, VSAM, or sequential.
Natural (DDM)	Name of the data definition module (DDM) for the secondary file. All fields in the secondary file must be in this DDM. This field defaults to the name of the secondary file, so you only need to enter a DDM if it is different from the secondary file name.
Secondary key	Name of the scrolling key used to read the secondary file in logical rather than physical sequence. This key must be defined as a descriptor, superdescriptor, or subdescriptor in the Predict file definition.

Field	Description
	Note: The secondary key is not applicable when the secondary file is a sequential work file.
Related by field	Name of the primary key used to couple the primary and secondary files.
Predict Relationships	
Select all	If this field is marked, the generated program performs file lookups (joins) on all files related to the secondary file in Predict. In each relationship, the cardinality of the secondary file must be N or CN. The cardinality of the related file must be 1 or C. The update constraint type must be R (restricted update). The delete constraint type can be blank or R. Only type N (Natural Construct) relationships are processed. Within the generated batch program, the relationship name is used as the view name. Specify the related file name when selecting fields in the WRITE-FIELDS user exits. To avoid having to make manual changes after generating the user exit, change the related file name to the corresponding relationship name before generation. The generated fields will then have the correct prefix value. **Note:** For DB2 users, type R (referential constraint) relationships are also processed.
Select specific	Names of up to four Predict relationships for the specified secondary file.

The following example shows the **Specify Parameters For Secondary File (1)** panel:



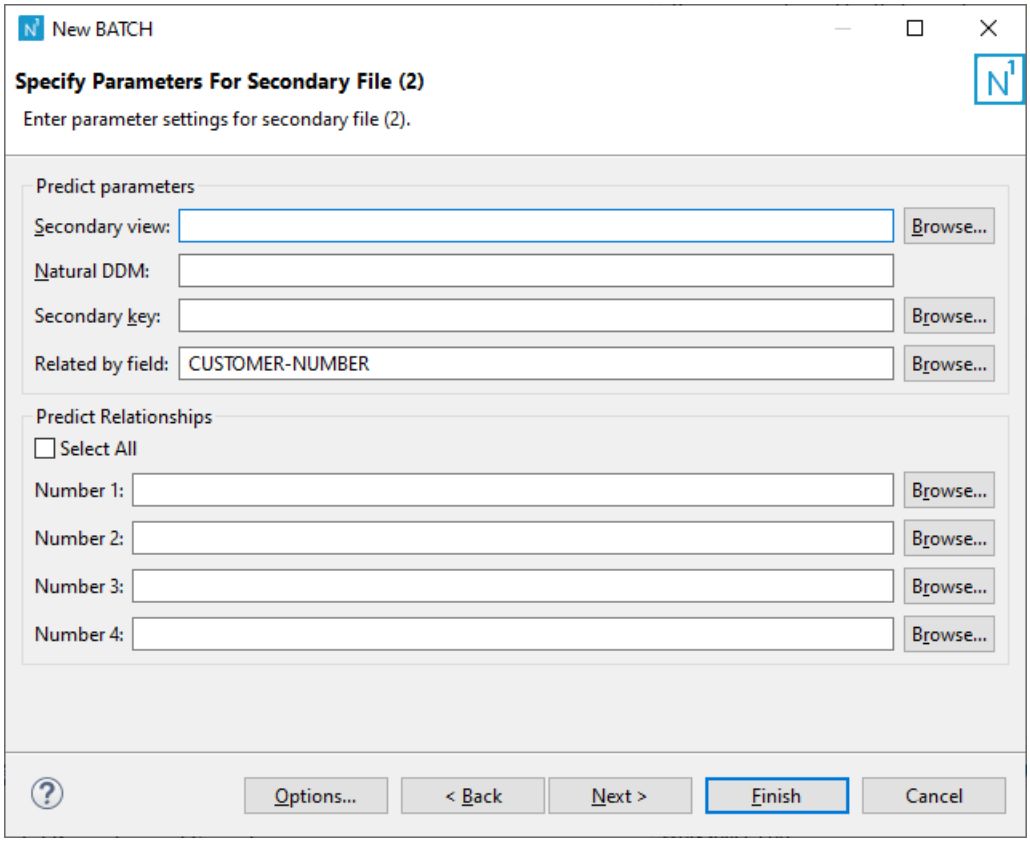
Specify Parameters for Secondary file (2)

You can specify a secondary file, which will be logically coupled (joined) to the primary file in your batch program. The secondary file must be normally related to the primary file with cardinality 1:N.

Panel reference

Field	Description
Predict parameters	
Secondary view	Name of the secondary view, which must be defined in Predict. The secondary file type can be Adabas, DB2, VSAM, or sequential.
Natural (DDM)	Name of the data definition module (DDM) for the secondary file. All fields in the secondary file must be in this DDM. This field defaults to the name of the secondary file, so you only need to enter a DDM if it is different from the secondary file name.
Secondary key	Name of the scrolling key used to read the secondary file in logical rather than physical sequence. This key must be defined as a descriptor, superdescriptor, or subdescriptor in the Predict file definition. **Note:** The secondary key is not applicable when the secondary file is a sequential work file.
Related by field	Name of the primary key used to couple the primary and secondary files.
Predict Relationships	
Select all	If this field is marked, the generated program performs file lookups (joins) on all files related to the secondary file in Predict. In each relationship, the cardinality of the secondary file must be N or CN. The cardinality of the related file must be 1 or C. The update constraint type must be R (restricted update). The delete constraint type can be blank or R. Only type N (Natural Construct) relationships are processed. Within the generated batch program, the relationship name is used as the view name. Specify the related file name when selecting fields in the WRITE-FIELDS user exits. To avoid having to make manual changes after generating the user exit, change the related file name to the corresponding relationship name before generation. The generated fields will then have the correct prefix value. **Note:** For DB2 users, type R (referential constraint) relationships are also processed.
Select specific	Names of up to four Predict relationships for the specified secondary file.

The following example shows the **Specify Parameters For Secondary File (2)** panel:



Specify Parameters for Tertiary file (1)

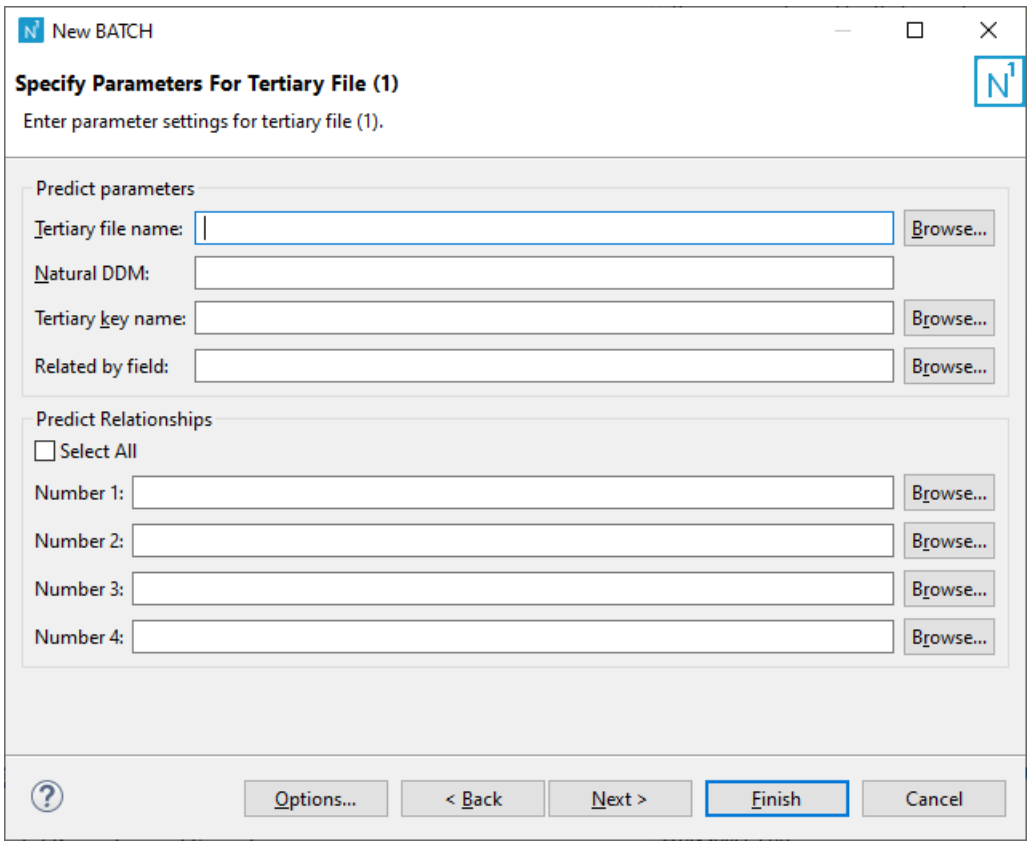
On this panel you can define a tertiary file, which will be logically coupled (joined) to the first secondary file.

Panel reference

Field	Description
Predict parameters	
Tertiary file name	Name of the tertiary view, which must be defined in Predict. The tertiary file type can be Adabas, DB2, VSAM, or sequential.
Natural (DDM)	Name of the data definition module (DDM) for the tertiary file. All fields in the secondary file must be in this DDM. This field defaults to the name of the tertiary file, so you only need to enter a DDM if it is different from the tertiary file name.
Tertiary key name	Name of the scrolling key used to read the tertiary file in logical rather than physical sequence. This key must be defined as a descriptor, superdescriptor, or subdescriptor in the Predict file definition. **Note:** The tertiary key is not applicable when the tertiary file is a sequential work file.

Field	Description
Related by field	Name of the secondary key used to couple the secondary and tertiary files.
Predict Relationships	
Select all	If this field is marked, the generated program performs file lookups (joins) on all files related to the tertiary file in Predict. In each relationship, the cardinality of the tertiary file must be N or CN. The cardinality of the related file must be 1 or C. The update constraint type must be R (restricted update). The delete constraint type can be blank or R. Only type N (Natural Construct) relationships are processed. Within the generated batch program, the relationship name is used as the view name. Specify the related file name when selecting fields in the WRITE-FIELDS user exits. To avoid having to make manual changes after generating the user exit, change the related file name to the corresponding relationship name before generation. The generated fields will then have the correct prefix value. **Note:** For DB2 users, type R (referential constraint) relationships are also processed.
Select specific	Names of up to four Predict relationships for the specified tertiary file.

The following example shows the **Specify Parameters For Tertiary File (1)** panel:



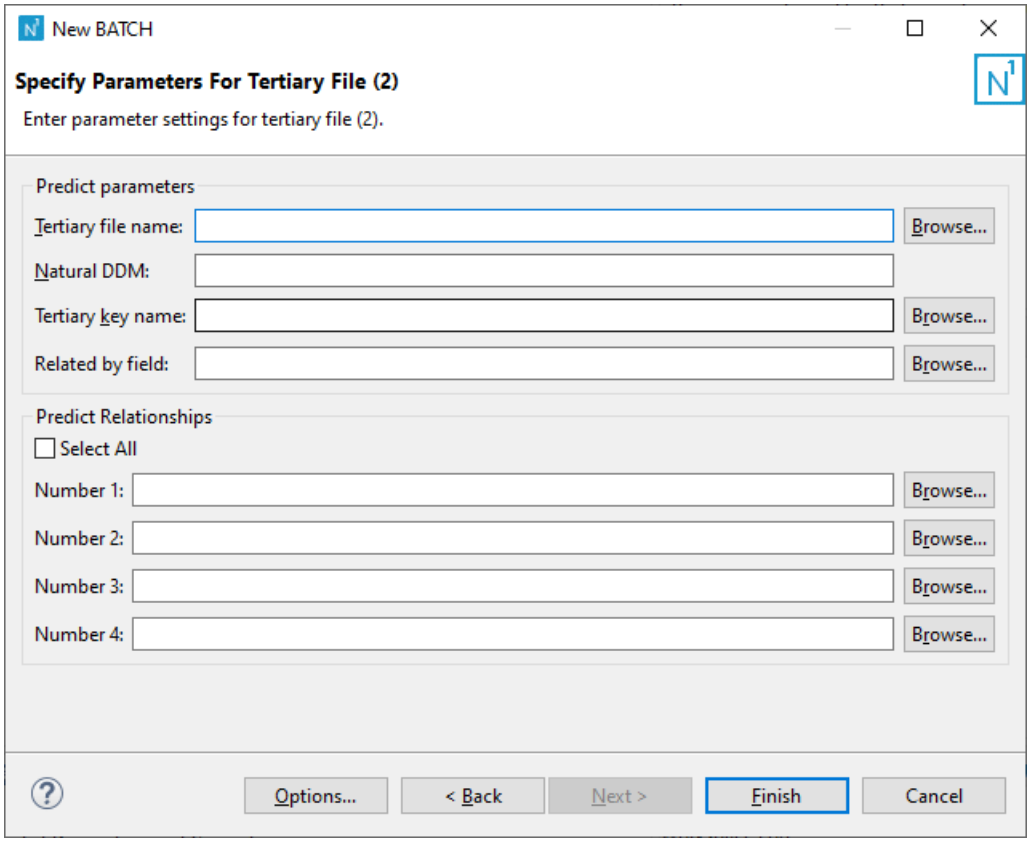
Specify Parameters for Tertiary file (2)

On this panel you can define a tertiary file, which will be logically coupled (joined) to the second secondary file.

Panel reference

Field	Description
Predict parameters	
Tertiary file name	Name of the tertiary view, which must be defined in Predict. The tertiary file type can be Adabas, DB2, VSAM, or sequential.
Natural (DDM)	Name of the data definition module (DDM) for the tertiary file. All fields in the secondary file must be in this DDM. This field defaults to the name of the tertiary file, so you only need to enter a DDM if it is different from the tertiary file name.
Tertiary key name	Name of the scrolling key used to read the tertiary file in logical rather than physical sequence. This key must be defined as a descriptor, superdescriptor, or subdescriptor in the Predict file definition. **Note:** The tertiary key is not applicable when the tertiary file is a sequential work file.
Related by field	Name of the secondary key used to couple the secondary and tertiary files.
Predict Relationships	
Select all	If this field is marked, the generated program performs file lookups (joins) on all files related to the tertiary file in Predict. In each relationship, the cardinality of the tertiary file must be N or CN. The cardinality of the related file must be 1 or C. The update constraint type must be R (restricted update). The delete constraint type can be blank or R. Only type N (Natural Construct) relationships are processed. Within the generated batch program, the relationship name is used as the view name. Specify the related file name when selecting fields in the WRITE-FIELDS user exits. To avoid having to make manual changes after generating the user exit, change the related file name to the corresponding relationship name before generation. The generated fields will then have the correct prefix value. **Note:** For DB2 users, type R (referential constraint) relationships are also processed.
Select specific	Names of up to four Predict relationships for the specified tertiary file.

The following example shows the **Specify Parameters For Tertiary File (2)** panel:



Browse/Browse-Select Wizards

This section describes the specification parameters for the Browse and Browse-Select series of wizards.

The following topics are covered:

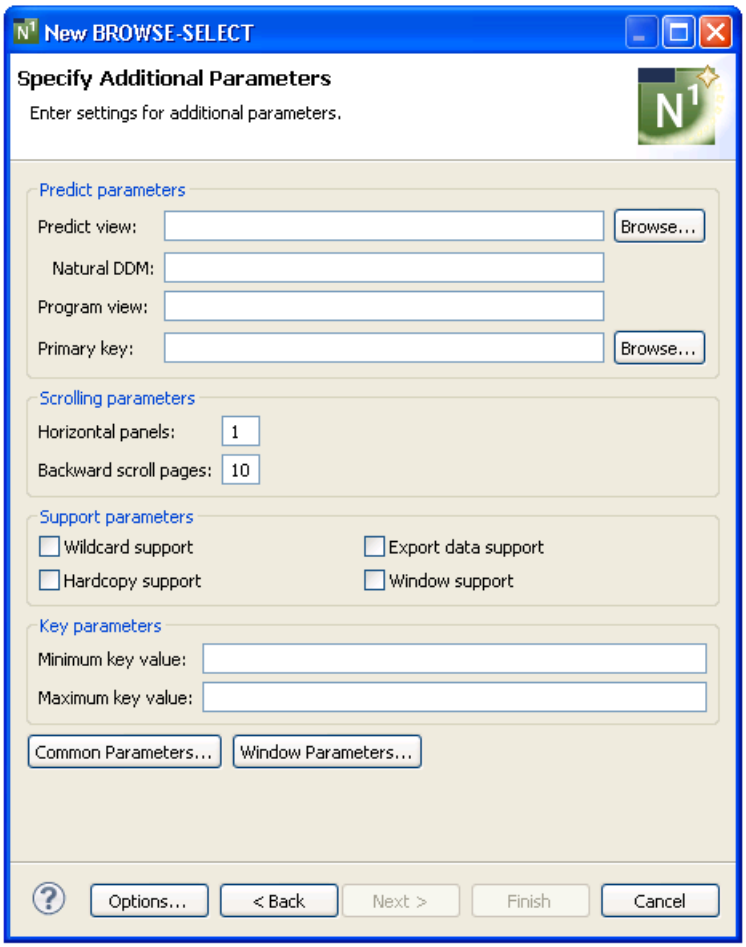
- Specify Standard Parameters
- Specify Additional Parameters
- Specify Map Details
- Specify Field Details
- Specify Restriction Parameters
- Specify Prefix Helproutine Parameters
- Specify #ACTION Parameters
- Specify Additional Subprogram Parameters

Note: The Browse-Select series of wizards is used for screen examples throughout this section.

Specify Standard Parameters

The **Specify Standard Parameters** panel is displayed when the wizard is invoked; it is similar for all Browse and Browse-Select wizards. For information about these parameters, see *Specify Standard Parameters*.

After specifying the standard parameters, select **Next** to display the **Specify Additional Parameters** panel. For example:



Specify Additional Parameters

To specify additional parameters

1 Define the following parameters:

Parameter	Description
Predict view	Name of the Predict view used by the generated browse module. Either type the name or select **Browse** to display the available views for selection. The view must be defined in Predict.
Primary key	Name of the primary key by which scrolling takes place. Either type the name or select **Browse** to display the available primary keys for selection. This key must be defined as a descriptor, superdescriptor, or subdescriptor in the Predict file definition. Keys containing MUs (multiple-valued fields) and PEs (periodic groups) are supported. If this key does not exist in the corresponding Predict file, a message is displayed. **Note:** For DB2 users, add the combination of fields as a superdescriptor in Predict if you want to use more than one field to determine the sort sequence of the records being browsed.

Optionally, you can:

Task	Procedure

Define the name of the data definition module (DDM) corresponding to the primary file.	Type the DDM name in **Natural DDM**. If this field is not specified, the DDM name defaults to the primary file name. The Predict definition of the primary file determines which fields are included in the DEFINE DATA section of your generated code. The format of the generated code in the DEFINE DATA section has the following structure:
	
	`1 Primary-file-name VIEW OF Data-definition-module`
	`  2 fields pulled from Predict of Primary-file-name`

Define the name of the view for the primary file for the generated module.	Type the primary file view name in **Program view**. This view must be defined in the LOCAL-DATA user exit or a local data area (LDA). If this field is not specified, a view is generated containing all fields in the Predict view. The MAX.OCCURS value in Predict determines how many occurrences of MU/PE fields are included on the panel.
Change the number of panels used for the generated module.	Type the number of panels in **Horizontal panels**. The default is 1 panel.
Change the maximum number of pages the generated module can scroll.	Type the maximum number of pages in **Backward scroll pages**. The default is 10, which indicates that users can scroll forward and backward within a 10-page range. If they scroll forward 11 pages, page 1 is forced out of the range and they cannot scroll back to it.

Task	Procedure
	Note: A Natural Construct-generated browse module does not allow backward scrolling over data that has not been previously scrolled through in a forward direction.
Enable wildcard processing in the generated module.	Select **Wildcard support**. Numeric key values are input into an alphanumeric field, which allows the user to enter "*", ">", or "<"
Enable records to be exported to a work file in addition to, or instead of, the screen.	Select **Export data support**. The work file can then be used in other environments and on other platforms (for example, in a PC spreadsheet application). To write data from the generated report to a work file, select the EXPORT-DATA user exit and define the parameters to export to the work file. The work file number and delimiter character (used to delimit fields on the report) can be customized for your site. **Note:** If you select this field and do not define the EXPORT-DATA user exit, a default WRITE WORK FILE statement that includes all fields in the view will be generated.
Enable the hardcopy facility in the generated module.	Select **Hardcopy support**.
Enable the module output to be displayed in a window, rather than a panel.	Select **Window support**. By default, the window size is adjusted to its content. The window is placed on the screen so that the field from which the user invoked it remains visible.
Define a starting value for the browse.	Type the starting value in **Minimum key value**. The combination of the minimum and maximum key values creates a logical window within the file. The program will not browse before or after these values. The minimum key value must be a constant. The specified constant is placed into a variable called #MIN-KEY-VALUE, which can be overridden in the START-OF-PROGRAM user exit.
Define an ending value for the browse.	Type the ending value in **Maximum key value**. The maximum key value must be a constant and greater than or equal to the minimum key value. The specified constant is placed into a variable called #MAX-KEY-VALUE, which can be overridden in the START-OF-PROGRAM exit. **Note:** You can set the minimum and maximum values as variables within user exit code. For example, if the first three characters of personnel ID represent the department code, you can restrict the browse to a specific department based on where the browse was called from or who was calling it. To do this, use the START-OF-PROGRAM user exit to look up and retrieve the current user's department code (assuming it is stored) and then use this information to populate a variable that overrides the #MIN-KEY-VALUE and #MAX-KEY-VALUE values (created when constants are populated through the specifications). If Smith belongs to department 555, for example, you can populate the minimum value

Task	Procedure
	with 555 and the maximum value with 55599999 to retrieve all data for department 555.
Define common parameters for the generated module, such as support for direct command processing, message numbers, and password checking.	Select **Common Parameters**. For information, see *Specify Common Parameters*.
Define window parameters for the generated module.	Select **Window Parameters**. For information, see *Change the Window Settings*.
Select generation options for the module(s).	Select **Options**. For information, see *Generation Options*.

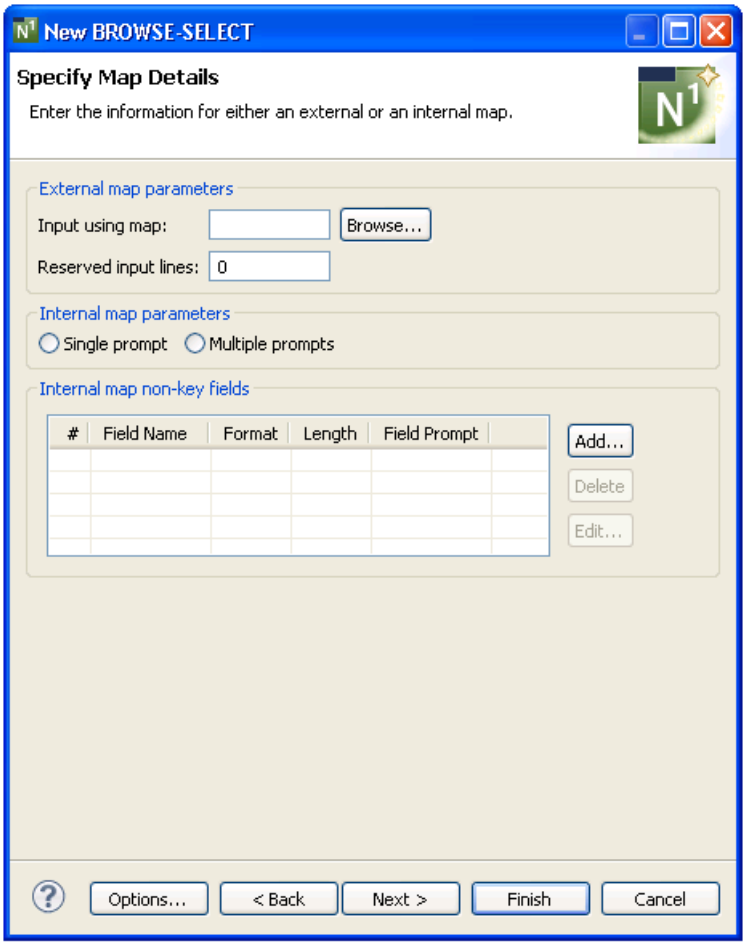
2 Select **Finish**.

The module is generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

Or:

Select **Next**.

The **Specify Map Details** panel is displayed. For example:



Specify Map Details

Optionally, you can define details for either an external or internal map.

To specify map details

1 Define the following optional parameters:

Parameter	Description
Input using map	Name of the layout map used for the generated module. Either type the name or select **Browse** to display the available maps for selection. If a map is specified, it should be a short map that is displayed at the bottom of the panel. The CDLAYSC1 layout map is supplied for Browse models. If you do not specify a layout map, Natural Construct places the input fields sequentially at the bottom of the panel. The map is included as part of the END OF PAGE processing to input values that control scrolling. The map must adhere to the following conventions:

Parameter	Description
	■ The map definition includes the #SCR-CV control variable. ■ The #KEY-CV control variable is defined for all input fields that are part of the browse key. ■ The input fields used to reposition the browse key, as well as any additional input fields, are defined within the #INPUT structure in the global data area (GDA) for the browse module. When more than one horizontal panel is required, use a different map for each panel. Include an asterisk (*) in the map name (for example, MYMAP*) and the asterisk will be replaced by the panel number during generation (for example, MYMAP1, MYMAP2, MYMAP3). If more than nine horizontal panels are used, the map name cannot exceed six bytes. To support an action/selection column for browse-select modules, include the column on the map as an array called #ACTIONS. Attach the #ACTION-CV control variable to #ACTIONS. To display a list of available actions on the generated panel, include the CDDIALDA.#KD-LINE1 and CDDIALDA.#KD-LINE2 variables on the map.
Reserved input lines	Number of lines reserved for input prompts (typically 1, 2, or 3).
Single prompt	Enables/disables a single prompt to be displayed for all fields (for example, Date:_____). This option applies when the key is a superdescriptor or redefined in Predict.
Multiple prompts	Enables/disables one prompt to be displayed for each field (for example, Year:____ Month:__ Day:__). This option applies when the key is a superdescriptor or redefined in Predict.
Internal map non-key fields	Up to eight additional input fields for the browse module. For information, see *Specify Field Details*.

2 Select **Finish**.

The module is generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

Or:

Select **Next**.

The following panel is displayed:

Wizard	Panel Displayed
Browse	*Specify Restriction Parameters*
Browse-Helpr and Browse-Subp	*Specify Additional Subprogram Parameters*
Browse-Select, Browse-Select-Helpr, and Browse-Select-Subp	*Specify #ACTION Parameters*

Specify Field Details

Optionally, you can specify up to eight additional input fields for a browse module. These fields are displayed at the bottom of the generated panel and allow the user to display more information. For example, you can create an additional input field called Detail (format L) to display additional record details. Users can select the **Detail** field to display the information.

Note: The additional input fields do not have to be in the Predict file definition.

This section covers the following topics:

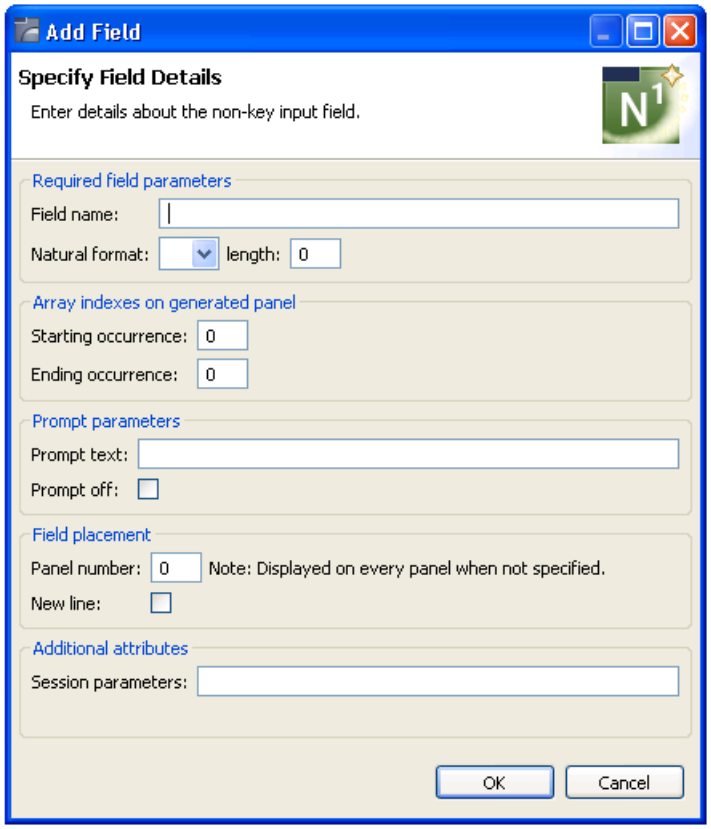
- Add a Non-Key Field
- Delete a Non-Key Field
- Edit a Non-Key Field

Add a Non-Key Field

To add a non-key field

1 Select **Add** on the **Specify Map Details** panel.

The **Specify Field Details** window is displayed. For example:



2 Define the following parameters for the additional field:

Parameter	Description
Field name	Name of the additional input field.
Natural format/length	Natural format and length for the field.
Starting occurrence	Starting occurrence of an array variable to place on the generated panel. **Note:** The Starting and Ending occurrence values define the range of occurrences of an array variable to place on the panel. If you specify a range with different first and last values, a single prompt precedes all elements in the range. (For multiple prompts, specify each occurrence separately.)
Ending occurrence	Ending occurrence of an array variable to place on the generated panel.
Prompt text	Text displayed for the field on the generated browse panel. Intensified text must be enclosed within angle (<>) brackets (or whatever attribute character is set for intensify).If you do not specify a field prompt, Natural Construct creates a prompt using the internal name of the input field.
Prompt off	Enables/disables the display of the prompt text.
Dynamic Attributes	Change the default dynamic attribute characters. For information, see *Change the Dynamic Attribute Characters*.

Parameter	Description
Panel number	Panel number for the field for the first dimension. If a panel number is not specified, the prompt is displayed on all panels.
New line	Enables/disables the display of the input field on a new line.
Session parameters	One or more session parameters for the additional input field, such as Attribute Definition (AD) or Edit Mask (EM). For example: `AD=I SG=ON EM='>'X HE='HELPR'`

3 Select **OK** to add the field.

Delete a Non-Key Field

To delete a non-key field

1 Select the field you want to delete on the **Specify Map Details** panel.

2 Select **Delete**.

The field is removed from the **Internal map non-key fields** table.

Edit a Non-Key Field

To edit a non-key field

1 Select the field you want to edit on the **Specify Map Details** panel.

2 Select **Edit**.

Or:

Double-click on the row in the **Internal map non-key fields** section.

The **Specify Field Details** window is displayed, showing the current settings for the field.

3 Edit the field settings.

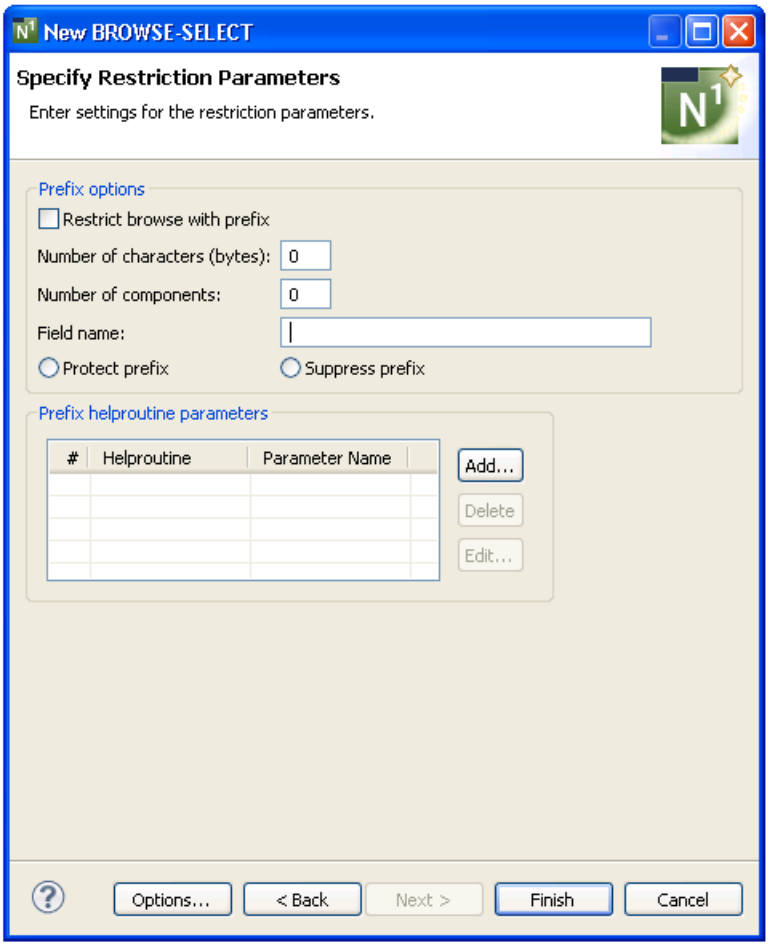
4 Select **OK** to save the changes.

Specify Restriction Parameters

For a browse program, you can optionally limit the generated module to only browse records prefixed by a global variable. If the prefix is a department code, for example, you can restrict the browse to only those orders prefixed by a particular department code by setting the value of the code as a function of a user ID and storing the value in the global data area.

For a browse helproutine or subprogram, you can limit the browse by passing the prefix portion of the key. To display only the lines for a particular order, for example, you can pass the order number (N6) to the subprogram and enter "N6" in the **Natural format** field on the **Specify Additional Subprogram Parameters** panel. On the **Specify Restriction Parameters** panel, mark the **Restrict browse with prefix** field and enter "6" in the **Number of characters (bytes)** field.

The following example displays the **Specify Restriction Parameters** panel for the Browse-Select wizard:



To specify restriction parameters

1 Define the following optional parameters:

Parameter	Description
Restrict browse with prefix	Enables/disables the restriction of the browse by prefix. When this option is selected, the browse is limited to values for which the primary key is prefixed by or equal to the specified value. **Note:** If you select this option, you must also specify either the number of characters or number of components to use as the prefix and provide a field name.
Number of characters (bytes)	Number of bytes of the primary key to use as the prefix.. **Note:** You can specify either the number of characters or the number of components, but not both.
Number of components	Number of compound key components to use as the prefix. You can then use the **Prefix helproutine parameters** options to assign the helproutine parameters. **Note:** You can specify either the number of characters or the number of components, but not both.
Field name	Name of the field containing the prefix value. The value must be a valid Natural field name. When generating a browse helproutine or subprogram, the prefix portion of the key is assumed to be equal to #PDA-KEY (i.e., the value of the key passed to the helproutine or subprogram). To override this default: 1. Enter the name of a variable in **Field name**. 2. Define the variable in the LOCAL-DATA user exit. 3. Assign a value to the field in the ASSIGN-PREFIX-VALUE user exit. The assigned value (instead of #PDA-KEY) will then be used as the value for the prefix portion of the key.
Protect prefix	Enables/disables the protection of the prefix portion of the primary key for the input field. When this option is selected, the prefix is displayed but cannot be changed. **Note:** To use this option, the primary key must be a superdescriptor, a compound IMS key, or redefined in Predict.
Suppress prefix	Enables/disables the display of the prefix portion of the primary key. When this option is selected, the prefix portion of the primary key is not displayed. **Note:** To use this option, the primary key must be a superdescriptor, a compound IMS key, or redefined in Predict.
Helproutine parameters	Up to two restriction helproutine parameters. For information, see *Specify Prefix Helproutine Parameters*. **Note:** This option only applies when **Number of components** is specified.

2 Select **Finish**.

The modules are generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

3 Save the generated modules.

At this point, you can:

- Use the NaturalONE Testing option to test a subprogram. For information, see *Test a Subprogram Directly* in *Application Testing*.
- Define user exits for the subprogram. For information, see *Defining User Exits*.
- Use NaturalONE functionality to upload all generated modules to the server.

Specify Prefix Helproutine Parameters

When **Number of components** is specified on the **Specify Restriction Parameters** panel, you can optionally attach a helproutine to the prefix of the primary key.

This section covers the following topics:

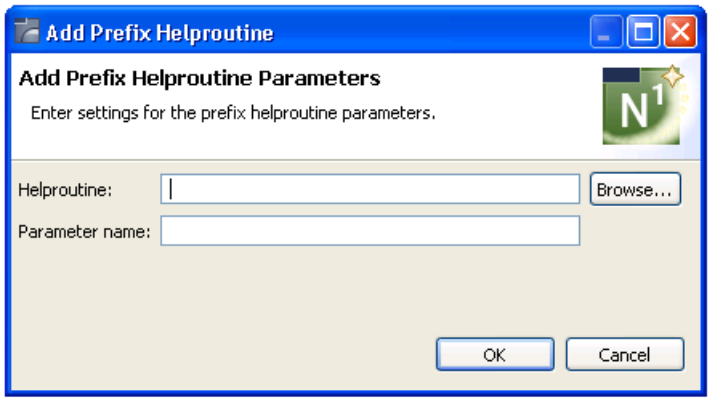
- Add a Prefix Helproutine Parameter
- Delete a Prefix Helproutine Parameter
- Edit a Prefix Helproutine Parameter

Add a Prefix Helproutine Parameter

To add a prefix helproutine parameter

1 Select **Add** on the **Specify Restriction Parameters** panel.

The **Add Prefix Helproutine Parameters** window is displayed. For example:



2 Define the following parameters for the additional field:

Parameter	Description
Helproutine	Name of the helproutine for the prefix. To attach a helproutine to the prefix of the primary key, enter the name of the helproutine in this field. You can specify a helproutine for each component.
Parameter name	Parameter for the helproutine for each component. Define the help parameters in the LOCAL-DATA user exit.

3 Select **OK** to add the helproutine.

Delete a Prefix Helproutine Parameter

To delete a prefix helproutine parameter

1 Select the helproutine you want to delete on the **Specify Restriction Parameters** panel.

2 Select **Delete**.

The helproutine is removed from the **Prefix helproutine parameters** table.

Edit a Prefix Helproutine Parameter

To edit a prefix helproutine parameter

1 Select the helproutine you want to delete on the **Specify Restriction Parameters** panel.

2 Select **Edit**.

The **Edit Prefix Helproutine** window is displayed, showing the current settings for the helproutine.

Or:

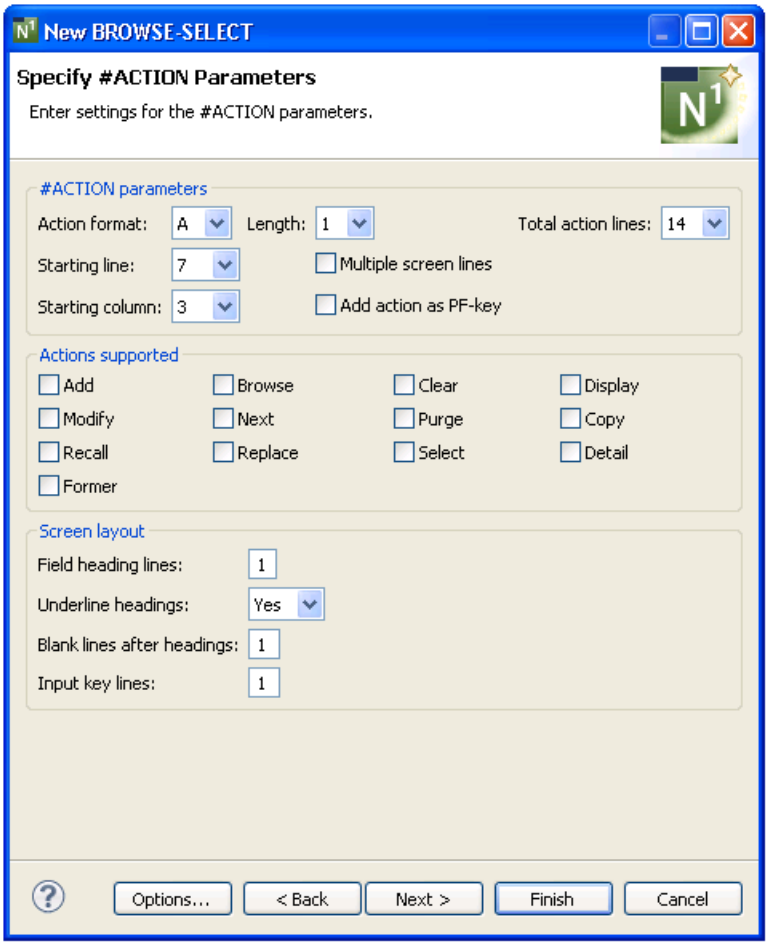
Double-click on the row in the **Prefix helproutine parameters** section.

3 Edit the helproutine settings.

4 Select **OK** to save the changes.

Specify #ACTION Parameters

For Browse-Select, Browse-Select-Helpr, and Browse-Select-Subp wizards, the**Specify #ACTION Parameters** panel is displayed after the **Specify Map Details** panel. This panel defines the characteristics of the action/selection field. For example:



To specify #ACTION parameters

1 Define the following optional parameters:

Parameter	Description
Action format/length	Natural format and length of the action/selection field. The default is A1.
Total action lines	Total number of action lines displayed on the generated panel. This number corresponds to the maximum number of database records. The default is 14 lines.

Parameter	Description
Starting line	Line number on which the action column begins. This number corresponds to the first line containing database information (after the panel and field heading lines). The default is 7.

Multiple screen lines	Determines whether each record requires more than one line. For example:
	`Address: Number-Street`
	`City, Province`
	`Country, Postal Code`

Starting column	Number of the column in which the action/selection fields are displayed (when not using an external map). This number determines the placement of the action/selection entries. The default is 3.
Add action as PF-key	Enables a PF-key for the Add action (by default, PF4). When generating a browse-select panel, you must decide how users will add records to a file: by entering "A" in the Action field or by pressing an Add PF-key. The first method is effective when adding records to existing files containing one or more records; the second method allows the user to select the Add PF-key while the cursor is positioned anywhere on the screen and add a record to an empty file.
Actions supported	Actions enabled for the generated browse-select panel. By default, no actions are supported.
Screen layout	The parameters in this section are used to build the screen layout when not using a predefined map.
Field heading lines	Number of field heading lines. The default is one line.
Underline headings	Determines whether field headings are displayed with a line under them. By default, field headings are underlined.
Blank lines after headings	Number of blank lines between the field headings and the data region. The default is one line.
Input key lines	Number of lines reserved at the bottom of the panel for input keys and additional fields. The default is one for input keys, plus the number of lines for additional input fields that begin on new lines. **Note:** Do not include the Direct Command line in the calculation of this value.

2 Select **Finish**.

The module is generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

Or:

3 Select **Next**.

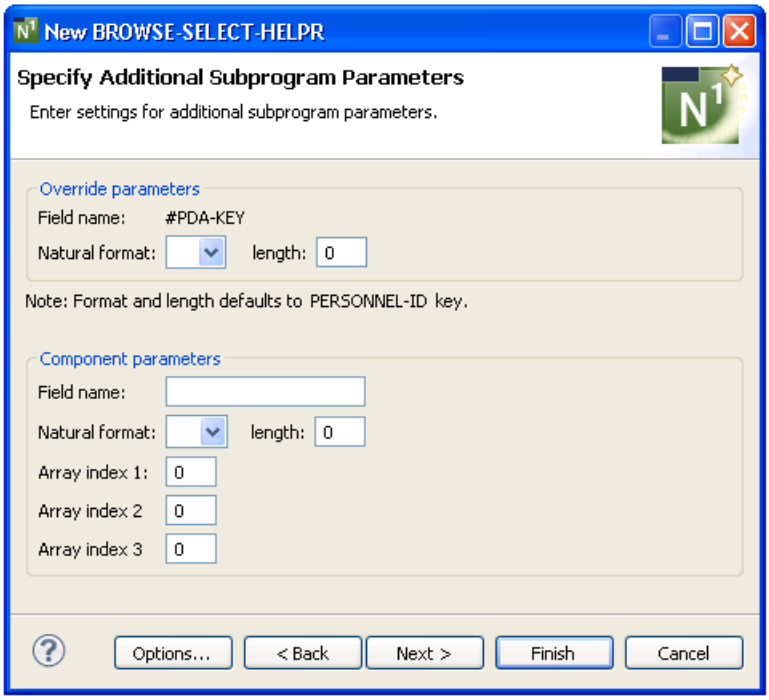
The following panel is displayed:

Wizard	Panel Displayed
Browse-Select	*Specify Restriction Parameters*
Browse-Select-Helpr or Browse-Select-Subp	*Specify Additional Subprogram Parameters*

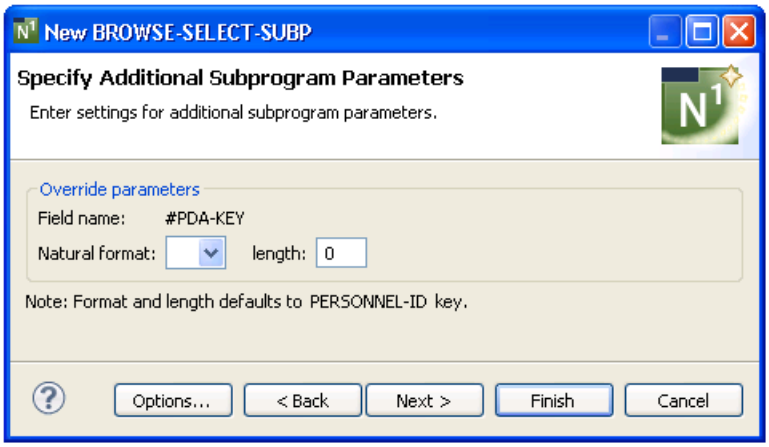
Specify Additional Subprogram Parameters

The **Specify Additional Subprogram Parameters** panel is displayed after the **Specify Map Details** panel for Browse-Helpr and Browse-Subp wizards and after the **Specify #ACTION Parameters** panel for Browse-Select-Helpr and Browse-Select-Subp wizards.

When generating a helproutine, the following information is displayed on the **Specify Additional Subprogram Parameters** panel:



When generating a subprogram, the following information is displayed on the **Specify Additional Subprogram Parameters** panel:



Use this panel to override the format and/or length of the passed parameter or pass an additional parameter to the helproutine. The key for the browse-select may differ from the key for the calling program. If the key differs, indicate the format and length of the passed key on this panel. Also indicate the name of any additional helproutine parameter, as well as its format and length.

Use the top portion of this panel to specify the format and length of the help field (if it is different from that of the primary browse key).

When generating a helproutine, use the bottom portion of this panel to specify additional parameters. If no additional parameters are specified, the generated helproutine only has one parameter (#PDA-KEY), which contains the contents of the input field to which the helproutine is attached. If the helproutine changes the value of #PDA-KEY, the altered value is displayed in the input field when the helproutine returns control to the INPUT statement.

To specify additional subprogram parameters

1 Define the following parameters:

Parameter	Description
Field name	Name of the primary key. By default, #PDA-KEY is displayed.
Natural format and length	Natural format and length of the passed field (if it is different from that of the key being browsed). This format becomes the format for the #PDA-KEY field.
Component parameters	
Field name	Name of the additional parameter.
Natural format and length	Natural format and length of the additional parameter. Any valid combination of format, length, and decimal positions under Natural is allowed.
Array index 1, 2, and 3	Array dimensions. To declare the additional parameter as an array, enter the array dimensions in the 1, 2, and 3 fields.

2 Select **Finish**.

The module is generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

Or:

Select **Next**.

The **Specify Restriction Parameters** panel is displayed. For information, see *Specify Restriction Parameters*.

Driver Wizard

This section describes the specification parameters for the Driver wizard. This wizard generates a module that executes a helproutine or subprogram for testing purposes. The Driver wizard generates an INPUT statement — you supply the parameters to execute the helproutine or subprogram. The wizard also generates headings and PF-key names according to the value of *Language.

This section covers the following topics:

- Specify Standard Parameters

Notes:

1. If Natural Construct does not find SYSERR text for the specified value of *Language at generation time, it uses the English text.
2. Because X-arrays must be materialized before they can be used in an INPUT statement, the Driver wizard materializes all X-arrays to one dimension. If other dimensions are required, manual changes must be made.

Specify Standard Parameters

The **Specify Standard Parameters** panel is the only specification panel for the Driver wizard.

To specify standard parameters

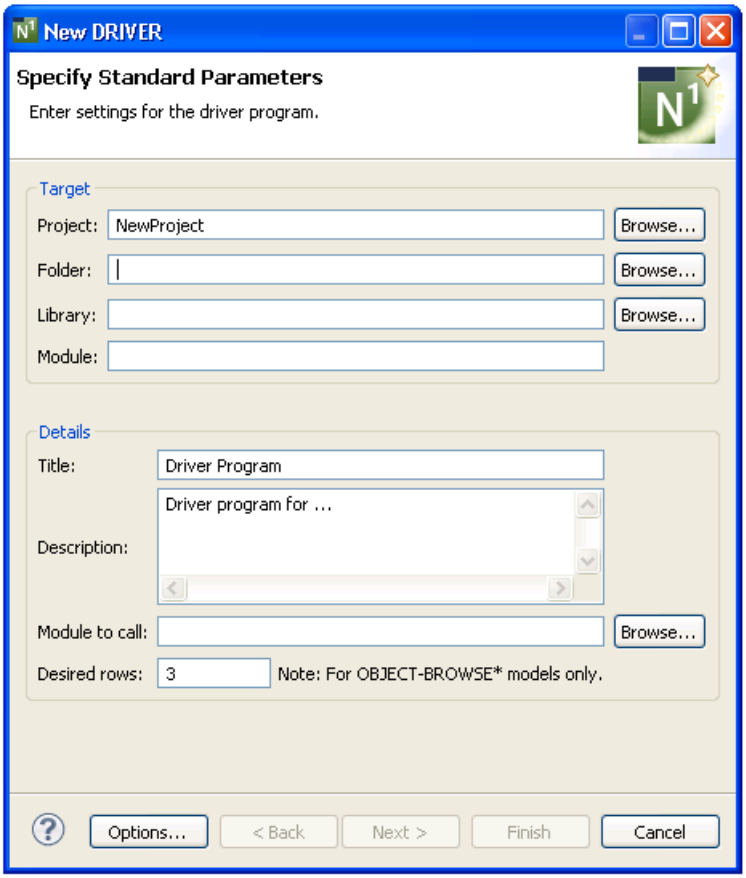
1 Open the context menu in the **Project Explorer** for the NaturalONE project in which you want to generate the modules.

Or:

Open the context menu in the **Project Explorer** for the library in which you want to generate the modules.

2 Select **Code Generation > New Using Construct Model > Driver**.

The **Specify Standard Parameters** panel is displayed. For example:



Many of the parameters on this panel are common to most wizards. For information, see *Specify Standard Parameters*.

3 Define the following parameter:

Parameter	Description
Module to call	Name of the helproutine or subprogram you want to execute using this driver program. Either type the name or select **Browse** to display the available modules for selection. A compiled version of the module must exist in the current library or in the steplib chain.

Optionally, you can:

Task	Procedure
Change the number of rows defined in the row array in the generated data PDA.	Type the new number in **Desired rows**. By default, three rows are defined. **Note:** This option is used when calling object browse subprograms generated using the supplied client generation wizard for NaturalONE (i.e., object-browse-n1). The wizard uses X-array technology at runtime to determine how many rows to generate in the data PDA.

4 Select **Finish**.

The driver program is generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

5 Save the generated module.

At this point, you can:

- Define user exits for the driver program. For information, see *Defining User Exits*.
- Use NaturalONE functionality to upload all generated modules to the server.

Maint Wizard

Natural Construct provides two alternatives to generate a maintenance process:

1. Use the Maint wizard. This wizard creates fast prototypes or simple maintenance processes that are temporary. The process is faster to implement because you need only create one or two Natural objects: the maintenance program generated by the Maint wizard and, optionally, a map. Conversely, because dialog and data are combined, the generated maintenance program is not as easy to modify as the subprograms generated by the Object wizards.
2. Use the Object wizards: *Object-Maint-Subp* or *Object-Maint-Enhanced-Subp* (which also generates the object PDA and restricted PDA) and, optionally, the *Object-Maint-Dialog* wizard. These wizards generate all the functionality needed for application development at the production level. The separation of dialog and data makes future changes to the maintenance process easier to implement.

The differences between the code generated by the two wizards include:

Maint Wizard	Object Wizards
Maintains one or two levels of files: primary and secondary.	Maintain up to four levels of files: primary, secondary, tertiary, and quaternary.
Supports one scrolling region.	Support multiple scrolling regions.
Does not support a link between scrolling regions on multiple panels.	Support a link between scrolling regions on multiple panels.
Does not provide automatic cursor repositioning after an error. (This functionality is available within user exits.)	Provide automatic cursor repositioning after an error.

This section describes the Maint wizard, which generates a program that maintains a file using a unique key and, optionally, a related secondary file. The Maint wizard generates the code necessary to maintain all the fields for an object, as well as scroll through the MU/PE fields of a primary file or the records of a secondary file.

The following topics are covered:

- Specify Standard Parameters
- Specify Additional Parameters
- Specify Additional Input Parameters
- Specify Secondary File Parameters

Note: By default, a maintenance program generated using the Maint wizard prompts users to press the Enter key to confirm a Purge action. If you specify a confirmation key other than Enter, the program will force confirmation of Add, Modify, and Purge actions. For a description of how to change the confirmation key, see *Confirmation Key Setup*, *Natural Construct Generation*.

Specify Standard Parameters

The **Specify Standard Parameters** panel is displayed when the wizard is invoked; it is similar for most wizards.

To specify standard parameters

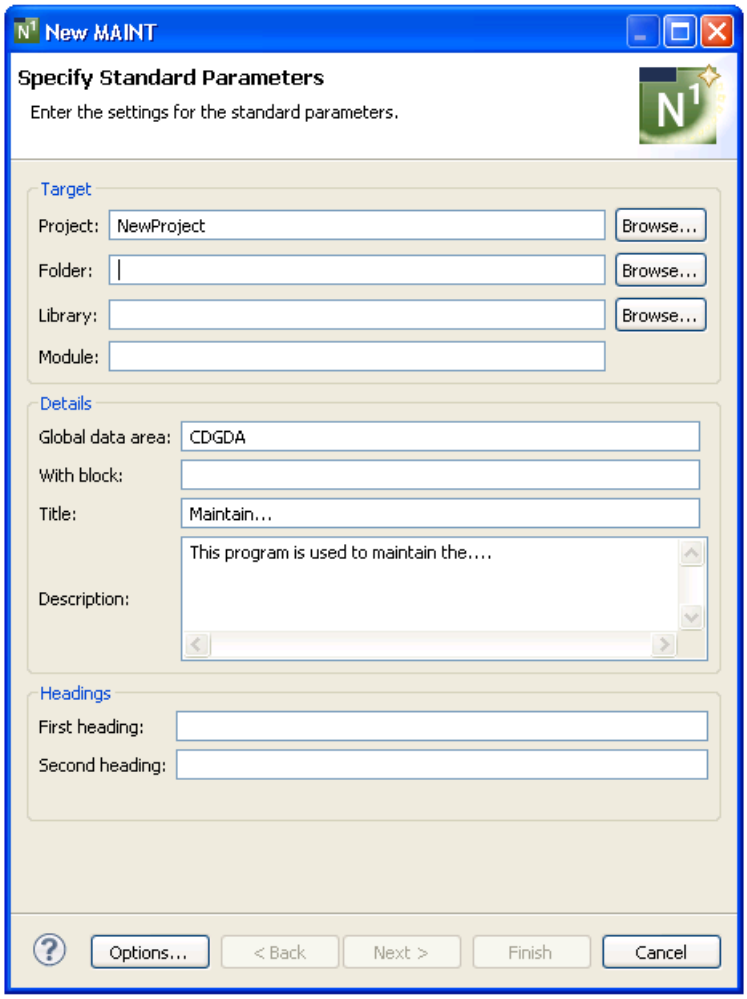
1 Open the context menu in the **Project Explorer** for the NaturalONE project in which you want to generate the modules.

   Or:

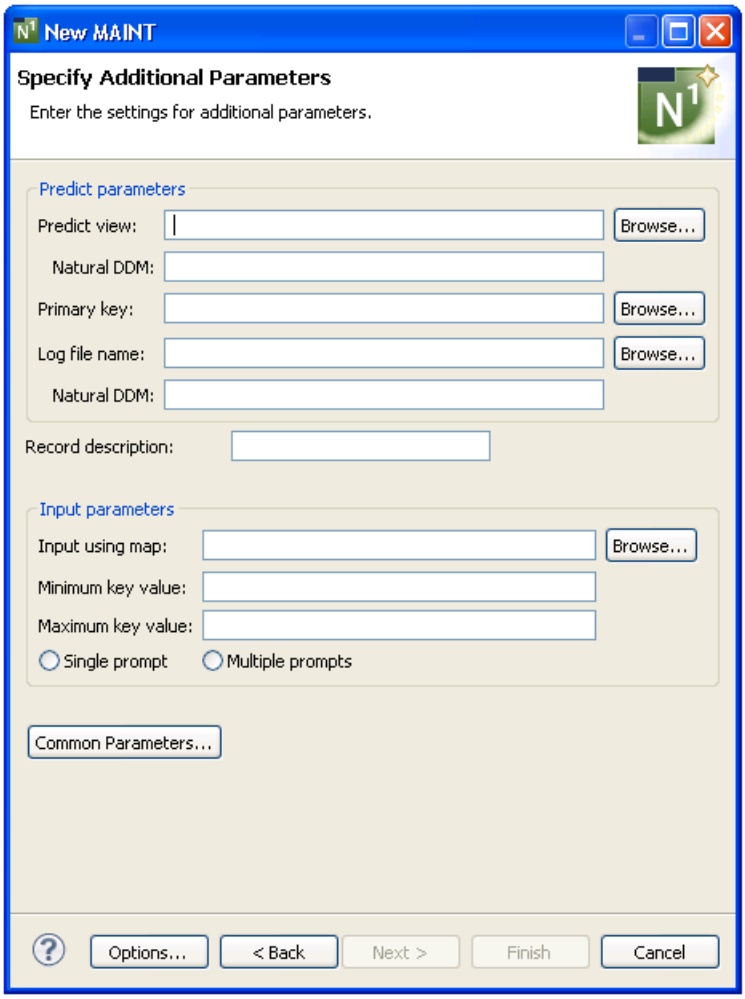
   Open the context menu in the **Project Explorer** for the library in which you want to generate the modules.

2 Select **Code Generation > New Using Construct Model > Maint**.

   The **Specify Standard Parameters** panel is displayed. For example:



For information about these parameters, see *Specify Standard Parameters*. After specifying the standard parameters, select **Next** to display the **Specify Additional Parameters** panel. For example:



Use this panel to define additional parameters for your maintenance program.

Specify Additional Parameters

To specify additional parameters

1 Define the following parameters:

Parameter	Description
Predict view	Name of the Predict view used by the generated browse module. Either type the name or select **Browse** to display the available views for selection. The view must be defined in Predict.

Parameter	Description
Primary key	Name of the primary key by which scrolling takes place. Either type the name or select **Browse** to display the available primary keys for selection. This key must be defined as a descriptor, superdescriptor, or subdescriptor in the Predict file definition. Keys containing MUs (multiple-valued fields) and PEs (periodic groups) are supported. If this key does not exist in the corresponding Predict file, a message is displayed. **Note:** For DB2 users, add the combination of fields as a superdescriptor in Predict if you want to use more than one field to determine the sort sequence of the records being browsed.

Optionally, you can:

Task	Procedure

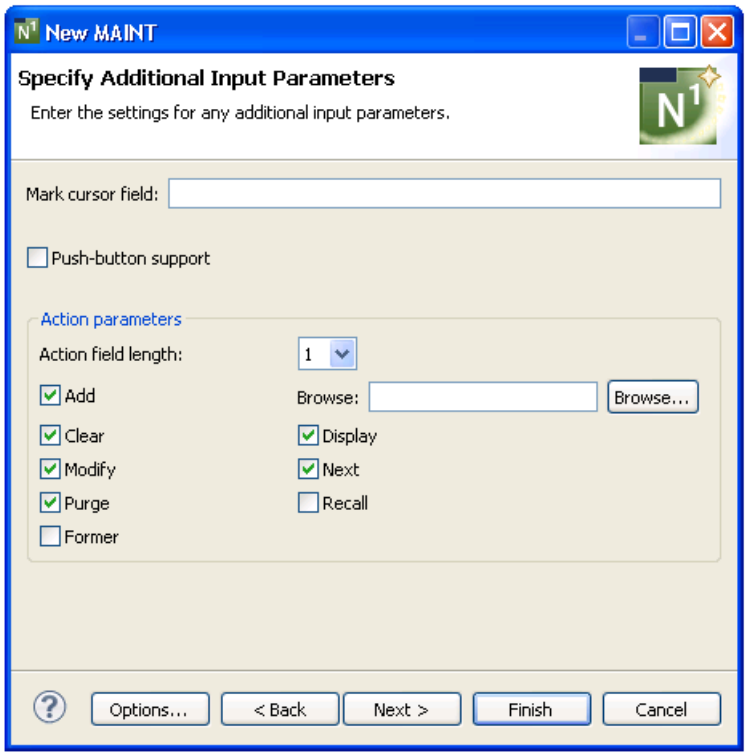
Define the name of the data definition module (DDM) corresponding to the primary file.	Type the DDM name in **Natural DDM**. If this field is not specified, the DDM name defaults to the primary file name. The Predict definition of the primary file determines which fields are included in the DEFINE DATA section of your generated code. The format of the generated code in the DEFINE DATA section has the following structure:
	
	`1 Primary-file-name VIEW OF Data-definition-module`
	`  2 fields pulled from Predict of Primary-file-name`

Provide the name of a log file to perform update logging for the generated maintenance program.	Type a view name in **Log file name** or select **Browse** to display the available views for selection. All fields in the primary file that have matching fields in the log file are written to the log records. Other fields, such as LOG-DATE and LOG-USER, must also appear in the log view.
Define the name of the data definition module (DDM) corresponding to the log file (when it is not the same as the log file name).	Type the DDM name in **Natural DDM**. If this field is not specified, the DDM name defaults to the log file name. The update log file must be a view of the log file DDM.
Provide a description of the record to be used in messages.	Type the description used in messages in **Record description**. By default, "Record" is displayed and messages are displayed in the form: "Record not found" and "Record displayed".
Provide the name of the layout map used for the generated maintenance program.	Type the name of the map in **Input using map** or select **Browse** to display the available maps for selection.
Define a starting value for the browse.	Type the starting value in **Minimum key value**. The combination of the minimum and maximum key values creates a logical window within the file. The program will not browse before or after these values.

Task	Procedure
	The minimum key value must be a constant. The specified constant is placed into a variable called #MIN-KEY-VALUE, which can be overridden in the START-OF-PROGRAM user exit.
Define an ending value for the browse.	Type the ending value in **Maximum key value**. The maximum key value must be a constant and greater than or equal to the minimum key value. The specified constant is placed into a variable called #MAX-KEY-VALUE, which can be overridden in the START-OF-PROGRAM exit. **Note:** You can set the minimum and maximum values as variables within user exit code. For example, if the first three characters of personnel ID represent the department code, you can restrict the browse to a specific department based on where the browse was called from or who was calling it. To do this, use the START-OF-PROGRAM user exit to look up and retrieve the current user's department code (assuming it is stored) and then use this information to populate a variable that overrides the #MIN-KEY-VALUE and #MAX-KEY-VALUE values (created when constants are populated through the specifications). If Smith belongs to department 555, for example, you can populate the minimum value with 555 and the maximum value with 55599999 to retrieve all data for department 555.
Display a single prompt for all fields on the generated panel.	Select **Single prompt**. This option enables/disables a single prompt to be displayed for all fields (for example, Date:____ __ __). It applies when the key is a superdescriptor or redefined in Predict.
Display multiple prompts for all fields on the generated panel.	Select **Multiple prompts**. This option enables/disables one prompt to be displayed for each field (for example, Year:____ Month:__ Day:__). It applies when the key is a superdescriptor or redefined in Predict.
Define common parameters for the generated module, such as support for direct command processing, message numbers, and password checking.	Select **Common Parameters**. For information, see *Specify Common Parameters*.
Select generation options for the module(s).	Select **Options**. For information, see *Generation Options*.

2 Select **Next**.

The **Specify Additional Input Parameters** panel is displayed. For example:



Use this panel to define any additional input parameters for your maintenance program.

Specify Additional Input Parameters

To specify additional input parameters

1 Define any or none of the additional parameters.

Using this panel, you can:

Task	Procedure
Define the field marked by default on the generated maintenance panel when an error occurs.	Type the name in **Mark cursor field**. To avoid ambiguity, fully qualify the field name with a structure name.
Present actions as cursor-sensitive push buttons.	Select **Push-button support**. Users can press the Tab key to move from action to action. For more information about implementing actions as push buttons, see the description of the #KD-LINES(*) variable in *Variables You Can Use with a Maint Model Map*, *Natural Construct Generation*.
Change the length of the field used for #ACTION names. By default, the field is 1 character in length.	Select another length in **Action field length** (possible lengths are 1, 2, or 3). For example, to use "DI" for the Display action, select "2" in this field.

Task	Procedure
Change the default actions generated for the maintenance program. By default, all actions except Recall and Former are selected.	Select or deselect any of the following actions: ■ Add (add a record to the file) ■ Clear (clear the specified record values from the panel) ■ Display (display the specified record) ■ Modify (modify the specified record) ■ Next (display the next record in the file) ■ Purge (removes the specified record from the file) ■ Recall (recall the values for the last record cleared from the panel following a Display, Modify, or Purge action) ■ Former (display the contents of the record having the next lower primary key value from the current key value; if no lower value exists, the `Start of Data` message is displayed)
Provide the name of the module used to perform the Browse action.	Select **Browse** for the **Browse** action field and select the name of the module.
Select generation options for the module(s).	Select **Options**. For information, see *Generation Options*.

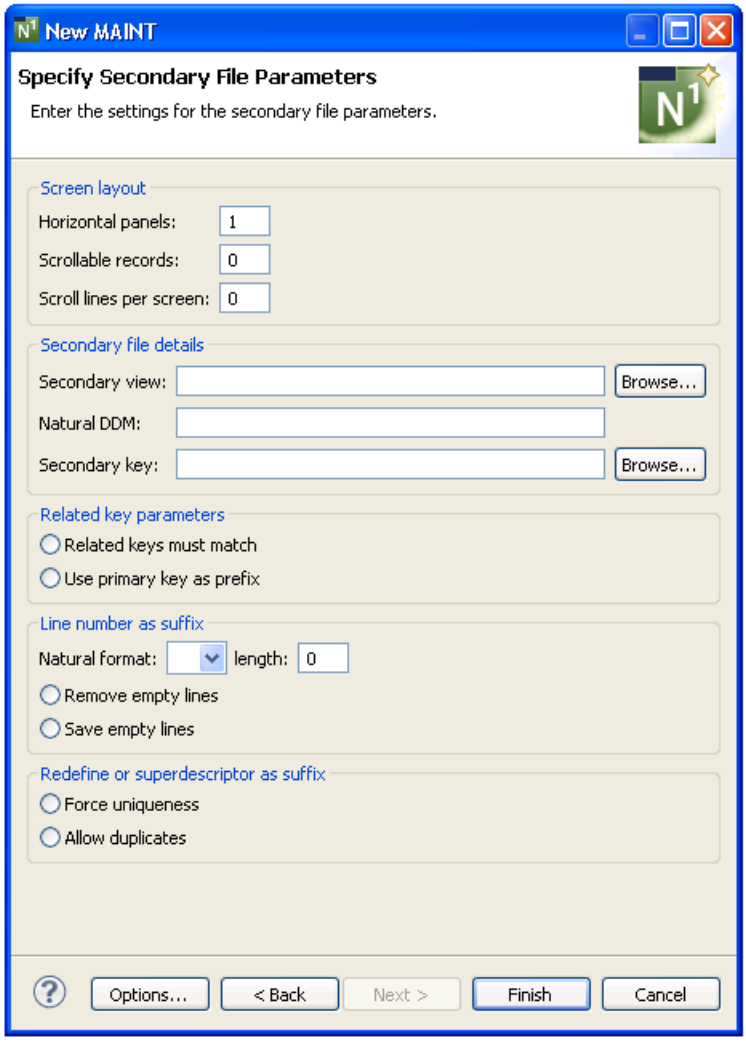
2 Select **Finish**.

The maintenance program is generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

Or:

Select **Next**.

The **Specify Secondary File Parameters** panel is displayed. For example:



Use this panel to define secondary parameters when the program maintains two files, periodic groups (PEs), multiple-valued fields (MUs), or uses more than one panel of input data.

Specify Secondary File Parameters

To specify secondary file parameters

1 Define any or none of the secondary file parameters.

Using this panel, you can:

Task	Procedure
Define the number of panels required to specify all data in the view. By default, "1" is displayed.	Type the number of panels in **Horizontal panels**. (The view may involve either one or two files.) The #PANEL value ranges from 1 to the number specified in this field. This option is used in conjunction with the **Input using map** field.
Define the maximum number of secondary file records that can be read or saved. By default, "0" is displayed.	Type the number of secondary file records in **Scrollable records**. If scrolling MU/PE fields in the primary file is supported, this value represents the highest value that may be scrolled. **Note:** 1. This value does not affect the number of MU/PE occurrences obtained; the MAX-OCCURS value in Predict determines this number. 2. If this option is specified, you must provide the name of a layout map in **Input using map** on the *Specify Additional Parameters* panel.
Define the number of MU/PE elements or secondary file records that can be displayed on the panel at one time. By default, "0" is displayed.	Type the number of elements or records in **Scroll lines per screen**. **Note:** If this option is specified, you must provide the name of a layout map in **Input using map** on the *Specify Additional Parameters* panel.
Specify the name of a Predict view that is coupled with the primary file for the maintenance program.	Type the name of the file in **Secondary view** or select **Browse** to display a window listing the existing files for selection. A file definition for the file must exist in Predict. If you specify a secondary view, you must also specify the maximum number of secondary file records that can be read or saved in **Scrollable records**, the number of scroll lines per panel in **Scroll lines per screen**, and select a secondary key in **Secondary key**.
Specify the name of the DDM (data definition module) for the secondary file.	Type the name of the DDM in **Natural DDM**. All fields in the secondary file must be in this DDM. **Note:** If you do not specify a secondary file DDM, this field defaults to the value in the **Secondary view** field.
Specify the name of the key in the secondary file that is related to the key in the primary file.	Type the name of the key in **Secondary key** or select **Browse** to display a window listing the existing keys for selection. The key can be a descriptor, superdescriptor, or subdescriptor.
Specify that the secondary file key must be identical to the primary file key.	Select **Related keys must match**.
Specify that the key of the primary file is a prefix of the secondary file key	Select **Use primary key as prefix**. If the primary file key is a prefix of the secondary file key, specify how the relationship between the two files is established by using either the Line

Task	Procedure
(secondary file records are always displayed in the secondary key order).	number as suffix options (**Remove empty lines** and **Save empty lines**) or the Redefine or superdescriptor as suffix options (**Force uniqueness** and **Allow duplicates**). The biggest difference between the two options is that for the Line number as suffix options, you do not need to enter the suffix value for the secondary key. Because the suffix value is a line number, the suffix is determined during each update session within the generated program. For all four options, the secondary file key can be either a descriptor or a superdescriptor. **Note:** The secondary key suffix may consist of one or more distinct fields that determine the sort sequence of the secondary file records. If this is the case, define a superdescriptor in Predict containing the fields that relate the secondary file to the primary file, followed by the fields that determine the sort order of the secondary records.
Specify the Natural format and length of the line number (N4, I2, for example).	Select the format in **Natural format** and type the length in **length**. The generated program assumes the secondary file key is made up of the primary file key value, plus a line number. The value of the suffix can be displayed on the panel, but cannot be modified.
Specify that the suffix components of the secondary file keys are renumbered (starting at 1) after an occurrence of the view is saved.	Select **Remove empty lines**.
Specify that the suffix components of the secondary file keys are not renumbered after an occurrence of the view is saved.	Select **Save empty lines**.
Use the redefinition of the secondary key field in **Predict** to determine the suffix (when the secondary key suffix is more than a line number).	Select **Force uniqueness**. The secondary key suffixes for each secondary file record should be modifiable when displayed on the map. The generated program also ensures a unique secondary file key. If the secondary key is a superdescriptor, place the trailing fields (beyond the primary key length) on the map.
Specify that the generated program does not check the secondary key for duplicates.	Select **Allow duplicates**.
Select generation options for the module(s).	Select **Options**. For information, see *Generation Options*.

2 Select **Finish**.

The maintenance program is generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

3 Save the generated module.

At this point, you can:

- Define user exits for the maintenance program. For information, see *Defining User Exits*.
- Use the NaturalONE functionality to test the program.
- Use NaturalONE functionality to upload the generated program to the server.

Menu Wizard

This section describes the specification parameters for the Menu wizard. This wizard generates a program that presents users with several choices in the form of a menu. The user enters a code for one of the choices to invoke a predefined function. You can also include additional fields on a menu, which may or may not require input.

This section covers the following topics:

- Specify Standard Parameters
- Specify Additional Parameters
- Define Menu Details
- Define Optional Input Parameters

Specify Standard Parameters

To specify standard parameters

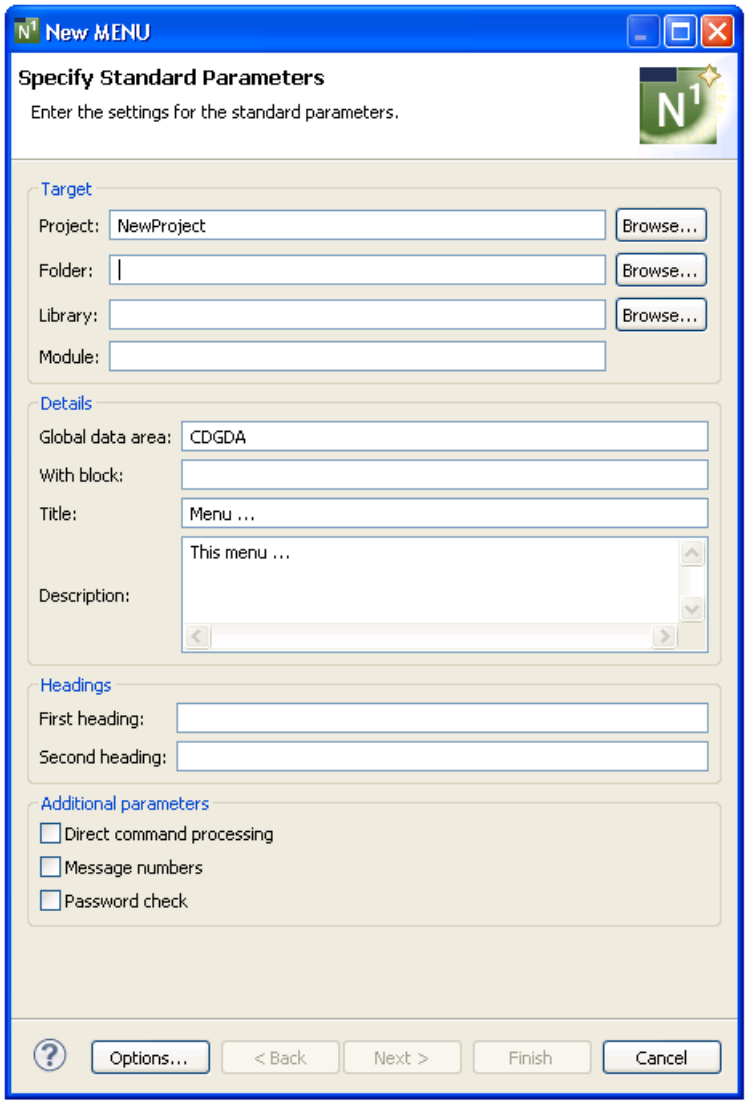
1 Open the context menu in the **Project Explorer** for the NaturalONE project in which you want to generate the modules.

Or:

Open the context menu in the **Project Explorer** for the library in which you want to generate the modules.

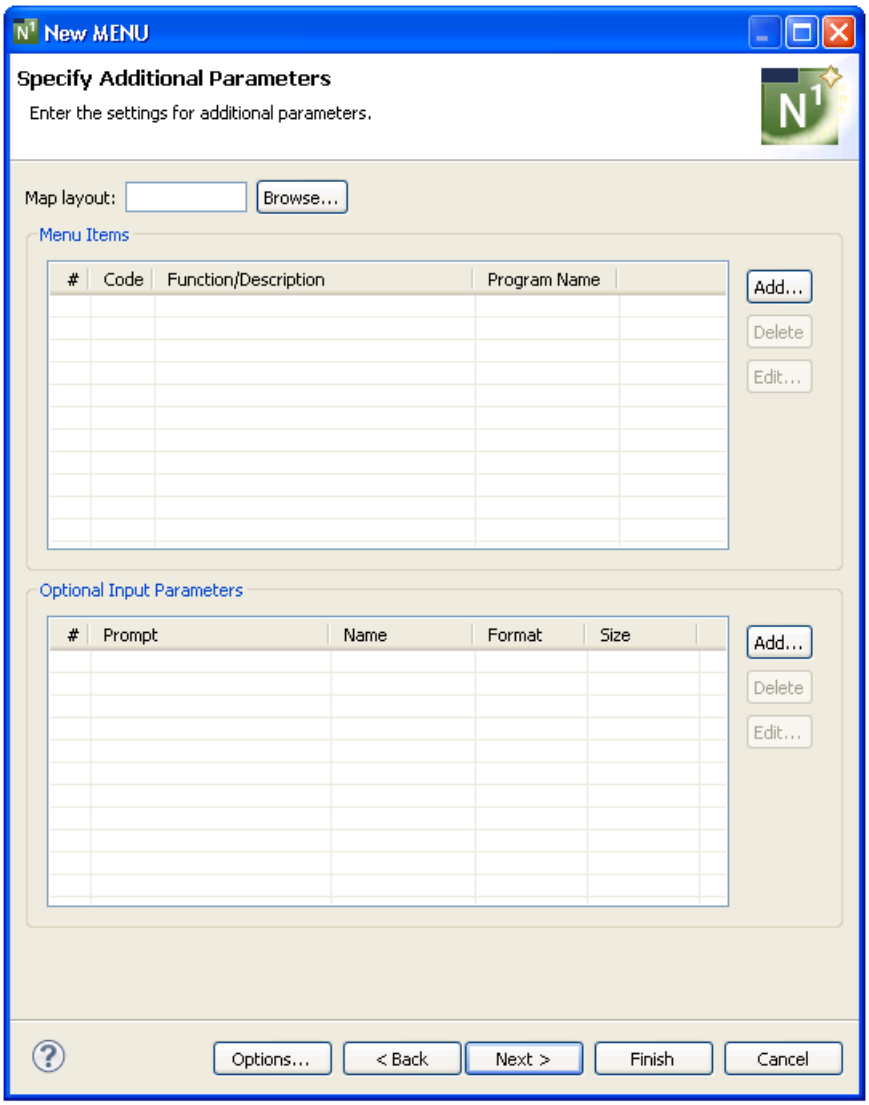
2 Select **Code Generation > New Using Construct Model > Menu**.

The **Specify Standard Parameters** panel is displayed. For example:



Many of the parameters on this panel are common to most wizards. For information, see *Specify Standard Parameters*. For information about the **Additional parameters** options, see *Specify Common Parameters*.

After specifying the standard parameters, select **Next** to display the **Specify Additional Parameters** panel. For example:



Use this panel to define parameters for the menu, such as the name of a map layout, the available codes and functions, and the names of the programs to FETCH (if entering a menu code invokes a program). Optionally, you can link up to four additional parameters to their associated menu functions.

Tip: Although a map layout is not required for a menu program, it can give a consistent, tailored appearance to your applications.

Specify Additional Parameters

To specify additional parameters

1 Select **Add** for **Menu items**.

The **Add Row** window is displayed. For information, see *Add a Row of Menu Items*.

Optionally, you can:

Task	Procedure
Provide the name of the layout map used for the generated menu program.	Type the name of the map in **Map layout** or select **Browse** to display the available maps for selection (.NSM file extension).
Link up to four additional parameters to their associated menu functions.	Select **Add** for **Optional input parameters**. For information, see *Define Optional Input Parameters*.

2 Select **Finish**.

The menu program is generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

3 Save the generated module.

At this point, you can:

- Define user exits for the menu program. For information, see *Defining User Exits*.
- Use NaturalONE functionality to upload all generated modules to the server.

Define Menu Details

This section describes how to define details for each row on your menu.

The following topics are covered:

- Add a Row of Menu Items
- Delete a Row of Menu Items

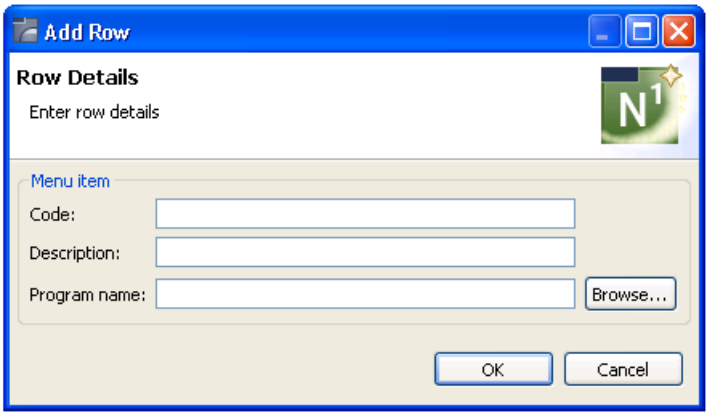
- Edit a Row of Menu Items

Add a Row of Menu Items

To add a row of menu items

1 Select **Add** for the **Menu items** table on the **Specify Additional Parameters** panel.

Add Row window is displayed. For example:



2 Define the following parameters for the menu row:

Field	Description
Code	One or two-character code users must enter to invoke the menu function.
Description	Function code descriptions displayed on the generated menu. If you are not using a layout map, you must provide a description for all codes specified in the **Code** field. The descriptions can be up to 45 characters in length.

Optionally, you can:

Task	Procedure
Provide the name of a program to FETCH (if entering a menu code invokes a program).	Type the name of the program in **Program name** or select **Browse** to display the available programs for selection (.NSP file extension). **Note:** If the menu function does not invoke a program, this field must be blank.

3 Select **OK** to add the row.

4 Perform steps 1, 2, and 3 until all menu rows have been added.

Delete a Row of Menu Items

To delete a row of menu items

1 Select the menu option you want to delete in **Menu items** on the **Specify Additional Parameters** panel.

2 Select **Delete**.

The row is removed from the **Menu items** table.

Edit a Row of Menu Items

To edit a row of menu items

1 Select the menu option you want to edit in **Menu items** on the **Specify Additional Parameters** panel.

2 Select **Edit**.

Or:

Double-click on the row in the **Menu items** table.

The **Edit Row** window is displayed, showing the current settings for the panel.

3 Edit the row settings.

4 Select **OK** to save the changes.

Define Optional Input Parameters

This section describes how to define optional input parameters, as well as how to link up to four parameters to their associated menu functions. (The majority of the menus will not use this feature.)

The following topics are covered:

- Add an Optional Input Parameter
- Delete an Optional Input Parameter

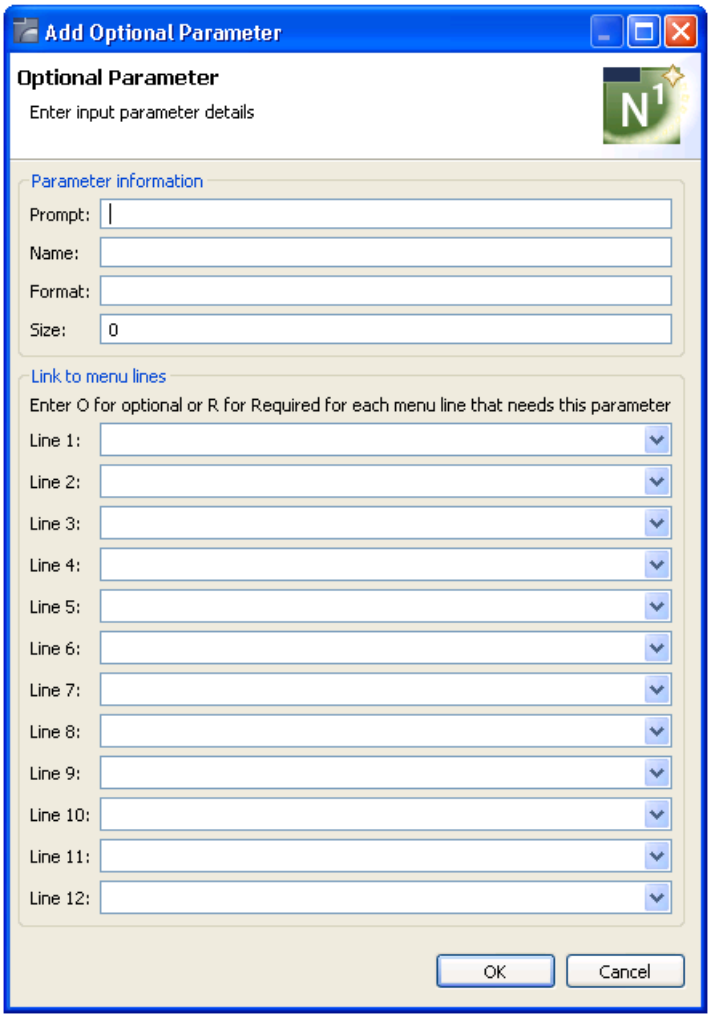
- Edit an Optional Input Parameter

Add an Optional Input Parameter

To add an optional input parameter

1 Select **Add** for the **Optional input parameters** table on the **Specify Additional Parameters** panel.

Add Optional Parameter window is displayed. For example:



2 Define the following fields for the optional parameter:

Field	Description
Prompt	Prompt displayed on the menu for the parameter.
Name	Name of a Natural internal variable to associate with the prompt. This variable checks the Required/Optional field to ensure that valid data is entered.
Format	Single-character alphabetical abbreviation for the Natural format of the specified variable (for example, N).
Size	Numeric length of the prompt.

Optionally, you can:

Task	Procedure
Provide the names of menu functions to be linked to the additional field.	Type the name of the field in Line *n* and select "R" (required) or "O" (optional). This link will provide a cross reference between the optional parameter and the menu functions.

3 Select **OK** to add the parameter.

Delete an Optional Input Parameter

To delete an optional input parameter

1 Select the parameter you want to delete in **Optional input parameters** on the **Specify Additional Parameters** panel.

2 Select **Delete**.

The row is removed from the **Optional input parameters** table.

Edit an Optional Input Parameter

To edit an optional input parameter

1 Select the parameter you want to edit in **Optional input parameters** on the **Specify Additional Parameters** panel.

2 Select **Edit**.

Or:

Double-click on the row in the **Optional input parameters** table.

The **Edit Optional Parameter** window is displayed, showing the current settings for the parameter.

3 Edit the parameter settings.

4 Select **OK** to save the changes.

Object-Browse-Dialog Wizard

This section describes the specification parameters for the Object-Browse-Dialog wizard. This wizard generates a character-based user interface to use with object-browse subprograms.

Because a browse module can be transformed into an object-browse subprogram and object-browse dialog program, and because the browse module has different PF-keys and actions and contains both UI and data access, you must consider how the Object-Browse-Dialog wizard works with a transformed object-browse subprogram versus one that was not transformed. If a browse module was transformed, the object-browse dialog program was generated automatically and there is no need to create one (but you can regenerate the dialog program). To differentiate between the two types in this section, these modules are referred to as a transformed object-browse dialog program versus an object-browse dialog program.

This section covers the following topics:

- Specify Standard Parameters
- Specify Additional Parameters
- Specify Specific Parameters

Note: For more information, refer to *Object-Browse-Dialog Model*, *Natural Construct Object Models*.

Specify Standard Parameters

The **Specify Standard Parameters** panel is displayed when the wizard is invoked; it is similar for all Object-Browse wizards.

To specify standard parameters

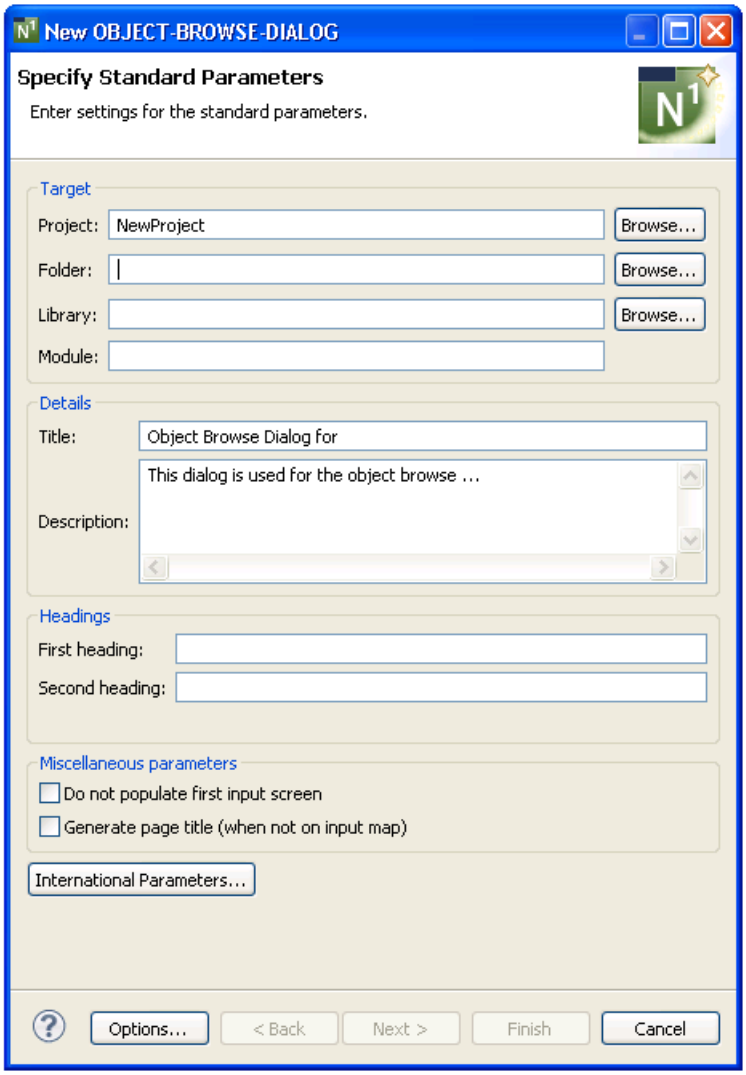
1 Open the context menu in the **Project Explorer** for the NaturalONE project in which you want to generate the modules.

   Or:

   Open the context menu in the **Project Explorer** for the library in which you want to generate the modules.

2 Select **Code Generation > New Using Construct Model > Object-Browse-Dialog**.

   The **Specify Standard Parameters** panel is displayed. For example:



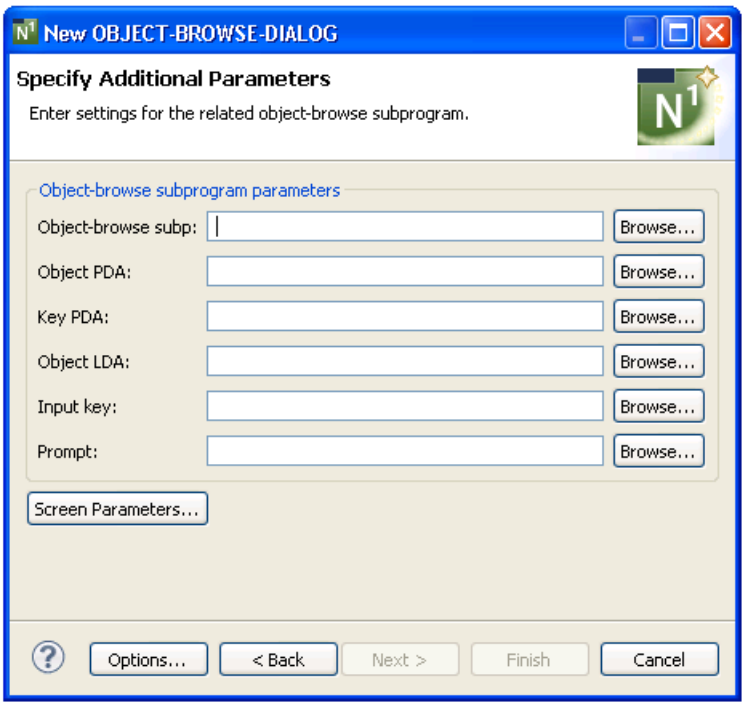
Many of the parameters on this panel are common to most wizards. For information, see *Specify Standard Parameters*. Optionally, you can use this panel to:

Task	Procedure
Suppress the population of fields with default data on the first input screen.	Select **Do not populate first input screen**. This option indicates whether default data is automatically entered into fields on the first input screen. It can be used to provide consistency between Natural Construct-generated browse and object-browse modules. By default, the first input screen is not populated for a transformed object-browse dialog program and is populated for a generated object-browse dialog program.
Generate a page title when it is not on the input map.	Select **Generate a page title (when not on input map)**. This option indicates whether to automatically code the page title when a map is not being used. By default, the page title is coded for a transformed

Task	Procedure
	object-browse dialog program and is not coded for a generated object-browse dialog program.
Define the language in which to display text on the generated panel(s).	Select **International Parameters**. For information, see *Specify International Parameters*.

3 Select **Next**.

The **Specify Additional Parameters** panel is displayed. For example:



Use this panel to define parameters for the related object-browse subprogram and, optionally, to define screen parameters. The generated object-browse dialog uses this subprogram to retrieve records for display.

Specify Additional Parameters

To specify additional parameters

1 Type the name of the subprogram used to retrieve records for display in **Object-browse subp** or select **Browse** to display the available subprograms for selection.

2 Type the name of the logical key by which scrolling takes place in **Input key** or select **Browse** to display the available logical keys for selection.

The specified key must be defined in the key PDA. For a transformed object-browse dialog program, the input key begins with A- (Ascending). To preserve the browse ascending and descending functionality, two keys are generated for the primary browse key: one that begins with A- and another that begins with D- (Descending). To expose this functionality to the end user, variables must be set in the START-OF-PROGRAM user exit.

Optionally, you can define the following parameters:

Parameter	Description
Object PDA	Object parameter data area (PDA) used by the specified object-browse subprogram. By default, the wizard will determine the name of the PDA based on the subprogram name. Alternatively, you can type the name or select **Browse** to display the available data areas for selection (.NSA file extension).
Key PDA	Name of the key PDA used by the specified object-browse subprogram. The key PDA is comprised of all fields that are components of the logical keys supported by the subprogram. By default, the wizard will determine the name of the PDA based on the subprogram name. Alternatively, you can type the name or select **Browse** to display the available data areas for selection (.NSA file extension).
Object LDA	Name of the object local data area (LDA) used by the generated dialog program. The object LDA contains the default field headings used when generating user exits. Either type the name or select **Browse** to display the available data areas for selection (.NSL file extension).
Prompt	Field name displayed for the input key on the generated panel. If a field name is not provided, the default name will be used. Either type the name or select **Browse** to display the available SYSERR numbers for selection. For information, see *Select a Message Number*.
Screen Parameters	Specify the language used to display text on the generated panels. See *Specify Screen Parameters*.

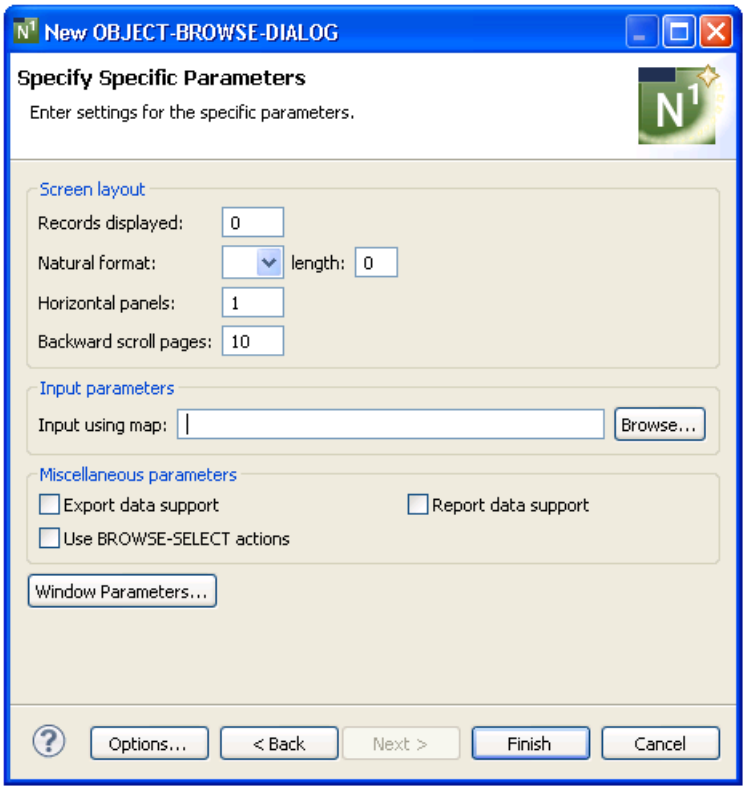
3 Select **Finish**.

The modules are generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

Or:

Select **Next**.

The **Specify Specific Parameters** panel is displayed. For example:



Use this panel to define the screen layout, map name, and support for exporting records to a work file or printer. You can also use this panel to change the style of actions used and the window settings.

Specify Specific Parameters

To specify specific parameters

1 Define the following optional parameters:

Parameter	Description
Records displayed	Number of records displayed on the screen at one time (by default, the generated dialog program displays 10 records at one time).
Natural format/length	Natural format and length for the selection column (for example, A1).
Horizontal panels	Number of horizontal panels used for the generated dialog program (by default, one panel is used).

Parameter	Description
Backward scroll pages	Maximum number of scroll pages within which users can scroll forward and backward (by default, 10 scroll pages).
Input using map	Name of the layout map used by the generated dialog program. Either type the name or select **Browse** to display the available maps for selection (.NSM file extension).
Export data support	If this option is selected, records are exported to a work file (instead of the screen).
Report data support	If this option is selected, records are exported to a local printer (instead of the screen).
Use BROWSE-SELECT actions	If this option is selected, the generated dialog program uses the same actions as those used by a BROWSE-SELECT-generated module. **Note:** This option is only relevant for transformed object-browse dialog programs and is not modifiable by this wizard.
Window Parameters	Specify window parameters. For information, see *Change the Window Settings*.

2 Select **Finish**.

The modules are generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

3 Save the generated modules.

At this point, you can:

- Define user exits for the dialog program. For information, see *Defining User Exits*.
- Use NaturalONE functionality to upload all generated modules to the server.

Object-Browse-Select-Subp Wizard

This section describes the specification parameters for the Object-Browse-Select-Subp wizard. This wizard generates a subprogram similar in functionality to a subprogram generated by the Browse-Select-Subp model. Both subprograms allow users to update multiple rows at one time. The primary difference between the two is that an object-browse-select subprogram can accommodate a client/server environment and you can use a subprogram proxy to access the generated code as a business service.

This section covers the following topics:

- Specify Standard Parameters

- Specify Additional Parameters

Note: For more information, refer to *Object-Browse-Select-Subp Model*, *Natural Construct Object Models*.

Specify Standard Parameters

The **Specify Standard Parameters** panel is displayed when the wizard is invoked; it is similar for all Object-Browse wizards.

To specify standard parameters

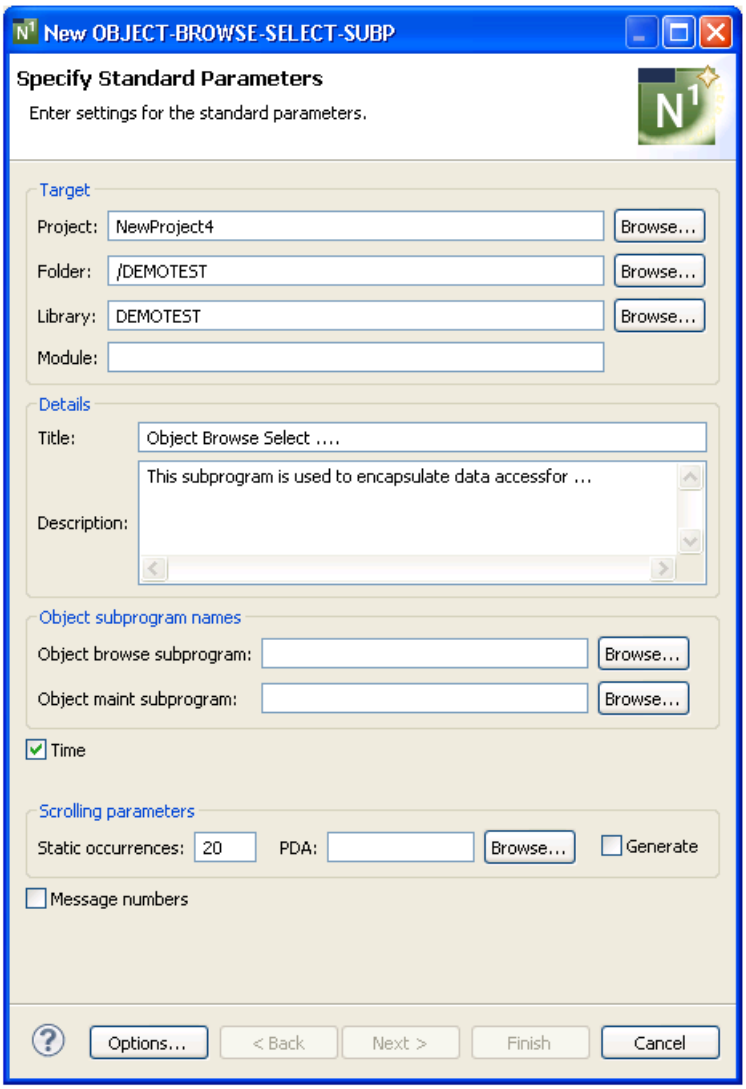
1 Open the context menu in the **Project Explorer** for the NaturalONE project in which you want to generate the modules.

   Or:

   Open the context menu in the **Project Explorer** for the library in which you want to generate the modules.

2 Select **Code Generation > New Using Construct Model > Object-Browse-Select-Subp**.

   The **Specify Standard Parameters** panel is displayed. For example:



Many of the parameters on this panel are common to most wizards. For information, see *Specify Standard Parameters*.

3 Define the following parameters:

Parameter	Description
Object-browse	Name of the subprogram used to browse this object. Either type the name of the subprogram or select **Browse** to display the available subprograms for selection. **Note:** The object-browse subprogram must be available in the current library.

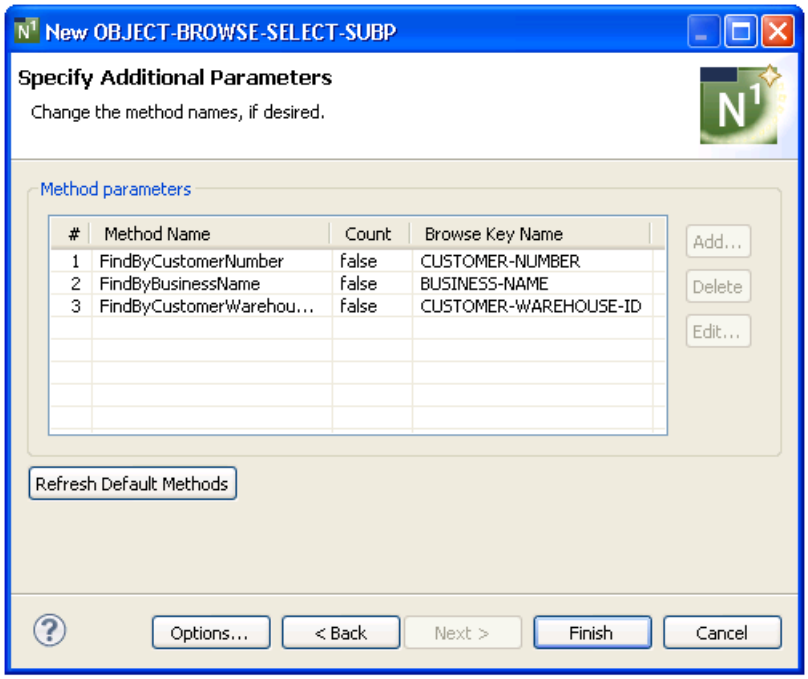
Parameter	Description
PDA	Name of the parameter data area associated with the static occurrences value for scrolling parameters. By default, the PDA name contains the first five characters of the module name and the number of static occurrences (for example, BCUSTS20). Either type the name or select **Browse** to display the available data areas for selection. **Note:** To generate the PDA, in addition to the object-browse-select subprogram, select **Generate**.

Optionally, you can:

Task	Procedure
Define an object-maintenance subprogram that will be used to maintain the object.	Type the name of the object-maint subprogram in **Object-maint** or select **Browse** to display a window listing the existing subprograms for selection. The subprogram must currently exist. The object-maintenance subprogram cannot process intra-object relationships. This allows the data presented to the client to be manageable and all data to be modifiable. **Note:** If you use an object-maint and an object-browse subprogram, both subprograms must use the same primary file.
Restrict the generation of code to time how long a business service takes to execute.	De-select **Time**. By default, this option is selected and code is generated to time how long a business service takes to execute. The result is returned in the business service message.
Change the number of rows processed and sent across the network at one time (by default, 20, unless the rows are extremely large).	Type the new number in **Static occurrences**. The PDA (parameter data area) associated with the static occurrences hard codes this value (which is used to identify the I/ value in the object-browse subprogram) in the object-browse-select subprogram. To identify the number of occurrences in this PDA, the default PDA name contains the first five characters of the module name and the number of static occurrences (for example, "BCUSTS20" when the static occurrences value is "20"). **Note:** If you change the number of static occurrences, you should also change this number in the name of the default PDA.
Use message numbers for all REINPUT and INPUT messages.	Select **Use message numbers**. When this option is selected, message numbers rather than message text will be used for all REINPUT and INPUT messages. **Note:** Use the same technique consistently throughout your application, since passing messages between modules using different techniques will not always produce the desired results.

4 Select **Next**.

The **Specify Additional Parameters** panel is displayed. For example:



This panel displays the names of the methods and browse keys used to determine the sort order for records returned by the object-browse subprogram (specified on the first wizard panel).

Specify Additional Parameters

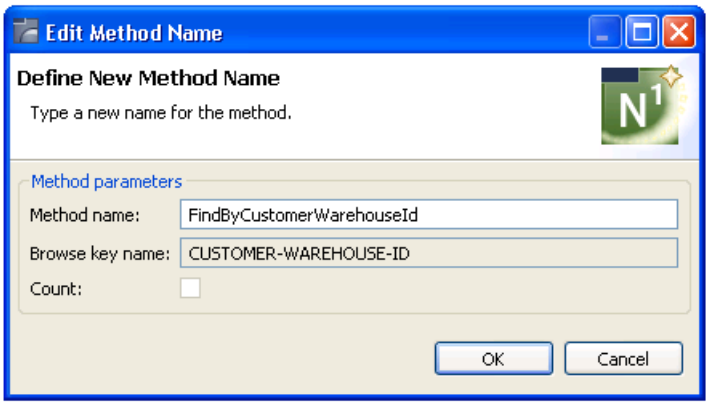
Use this panel to rename the default methods, if desired.

To rename the default methods

1 Select the method you want to rename in the **Method parameters** section.

2 Select **Edit**.

The **Edit Method Name** window is displayed. For example:



3 Change the name of the method in **Method name**.

4 Select **OK**.

The new name is displayed in the **Method parameters** section.

5 Select **Finish**.

The modules are generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

6 Save the generated modules.

At this point, you can:

- Use the NaturalONE Testing option to test a subprogram. For information, see *Test a Subprogram Directly* in *Application Testing*.
- Define user exits for the subprogram. For information, see ***Defining User Exits***.
- Use NaturalONE functionality to upload all generated modules to the server.

Object-Browse-Subp Wizard

This section describes the specification parameters for the Object-Browse-Subp wizard. This wizard generates the browse subprogram for an object, as well as three parameter data areas:

- Object PDA (defines the returned row data)
- Key PDA (defines the search key values)
- Restricted PDA (defines private data used internally by the browse object to maintain context)

This section covers the following topics:

- Specify Standard Parameters
- Specify Additional Parameters

- Specify Key Details
- Specify Logical Key Components

Note: For more information, refer to *Object-Browse-Subp Model*, *Natural Construct Object Models*.

Specify Standard Parameters

The **Specify Standard Parameters** panel is displayed when the wizard is invoked; it is similar for all Object-Browse wizards.

To specify standard parameters

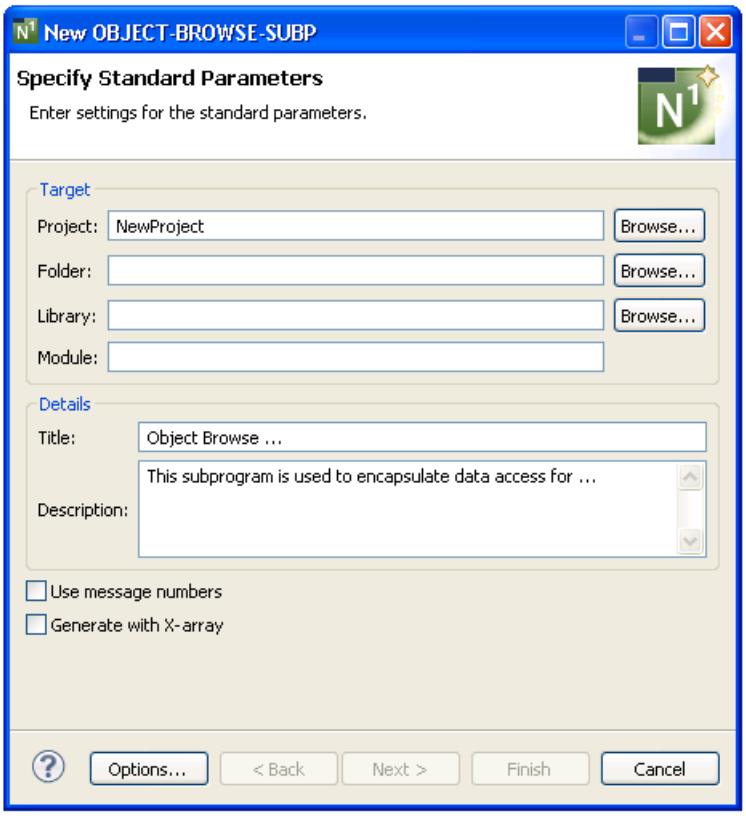
1 Open the context menu in the **Project Explorer** for the NaturalONE project in which you want to generate the modules.

   Or:

   Open the context menu in the **Project Explorer** for the library in which you want to generate the modules.

2 Select **Code Generation > New Using Construct Model > Object-Browse-Subp**.

   The **Specify Standard Parameters** panel is displayed. For example:

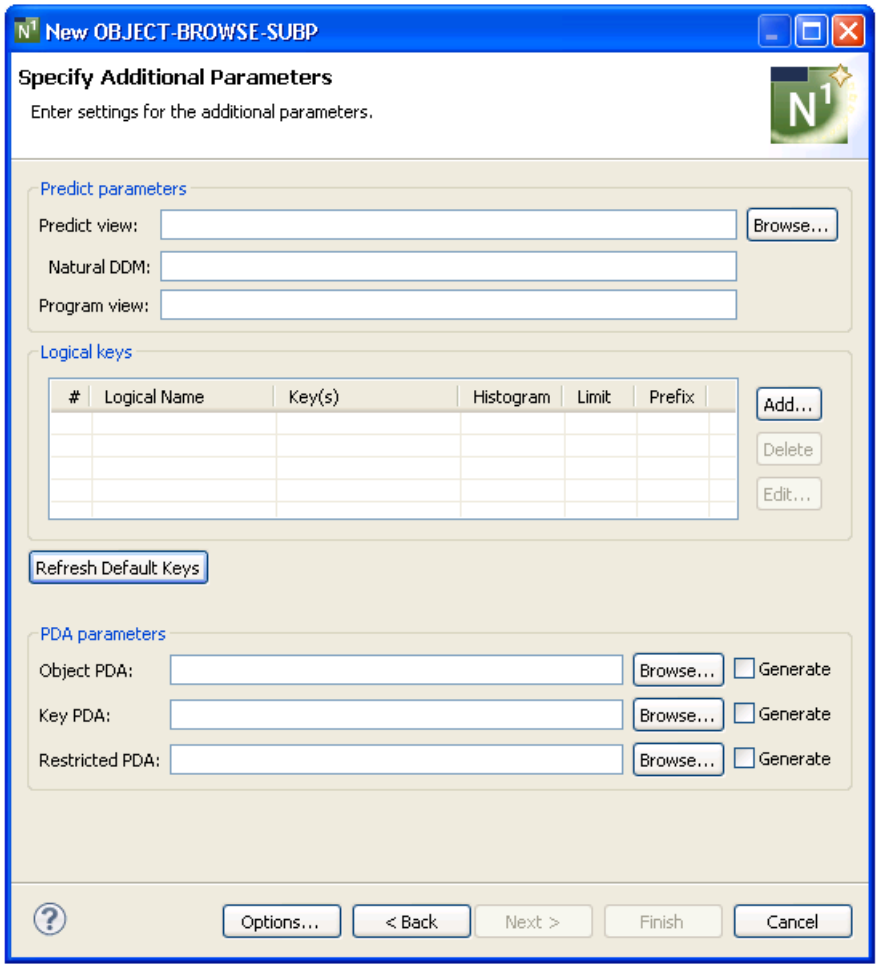


Many of the parameters on this panel are common to most wizards. For information, see *Specify Standard Parameters*. Optionally, you can:

Task	Procedure
Use message numbers for all REINPUT and INPUT messages.	Select **Use message numbers**. When this option is selected, message numbers rather than message text will be used for all REINPUT and INPUT messages. **Note:** Use the same technique consistently throughout your application, since passing messages between modules using different techniques will not always produce the desired results.
Generate the object (row) PDA with X-array support.	Select **Generate with X-array**. When this option is selected, the object PDA will be generated with (1:*) declarations instead of (1:V) for top-level rows; any arrays nested within top-level rows will be generated as defined in Predict.

3 Select **Next**.

The **Specify Additional Parameters** panel is displayed. For example:



Use this panel to define additional parameters for your object-browse subprogram.

Specify Additional Parameters

To specify additional parameters

1 Define the following parameters:

Parameter	Description	Required/Optional/Conditional
Predict view	Name of the Predict view used by the generated subprogram. The view must be defined in Predict. Either type the name or select **Browse** to display the available views for selection.	Required
Natural DDM	Name of the data definition module (DDM) corresponding to the primary file. If this field is not specified, the DDM name defaults to the primary file name. The Predict definition of the primary file determines which fields are included in the DEFINE DATA section of the generated code. The format of	Optional

Parameter	Description	Required/Optional/Conditional

	the generated code in the DEFINE DATA section has the following structure:
	
	`1 primary-file-name VIEW ↩`
	`OF data-definition-module`
	`  2 fields pulled from Predict ↩`
	`of primary-file-name`	

Program view	View name of the primary file for the generated subprogram. This view must be defined in the LOCAL-DATA user exit or a local data area (LDA). If this field is not specified, a view is generated containing all fields in the Predict view. The MAX.OCCURS value in Predict determines how many occurrences of MU/PE fields are included on the panel.	Optional
Logical keys	Up to six logical keys to determine the sort order for records returned by the object-browse subprogram. For information, see ***Specify Key Details***.	Optional
Refresh Default Keys	Retrieves the default key parameters for the specified Predict view and lists them in the **Logical keys** table.	Optional
Object PDA	Object parameter data area (PDA) that defines the rows returned to the object-browse subprogram and the columns within each row. Either type the name or select **Browse** to display the available PDAs for selection. Alternatively, you can select **Generate** to have the data area generated by the wizard. The generated object PDA contains one column for each field defined in the specified Predict view (as well as additional columns). You can remove any fields that are not components of the primary key. **Note:** When creating a new specification, this field is filled in by default with the first five bytes of the subprogram name, plus the suffix "ROW".	Required
Key PDA	Key PDA that contains all of the components contained in the logical keys, as well as a unique ID field. Either type the name or select **Browse** to display the available PDAs for selection. Alternatively, you can select **Generate** to have the data area generated by the wizard. **Note:** When creating a new specification, this field is filled in by default with the first five bytes of the subprogram name, plus the suffix "KEY".	Required

Parameter	Description	Required/Optional/Conditional
Restricted PDA	Restricted PDA that stores data, such as the last sort key, the last starting value, the last row returned, etc. so that the next set of consecutive records is returned to the caller. Either type the name or select **Browse** to display the available PDAs for selection. Alternatively, you can select **Generate** to have the data area generated by the wizard. **Note:** 1. The contents of this data area should not be altered by the calling module. 2. When creating a new specification, this field is filled in by default with the first five bytes of the subprogram name, plus the suffix "PRI".	Required

2 Select **Finish**.

The modules are generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

3 Save the generated modules.

At this point, you can:

- Use the NaturalONE Testing option to test a subprogram. For information, see *Test a Subprogram Directly* in *Application Testing*.
- Define user exits for the subprogram. For information, see ***Defining User Exits***.
- Use NaturalONE functionality to upload all generated modules to the server.

Specify Key Details

Optionally, you can specify up to six logical keys to determine the sort order for records returned by the object-browse subprogram. The calling program indicates the sort order by assigning CD-BRPDA.SORT-KEY. If a sort key value is not assigned, the first logical key is used as the default.

The logical key names can map to as many as five components. If a logical key contains only one component, the logical key name is optional. If you do not specify a logical key name, this field defaults to the name of the key component.

If the key field contains MU or PE fields, the rows returned also contain an index value that identifies which occurrence of the MU/PE field satisfies the Read condition.

This section covers the following topics:

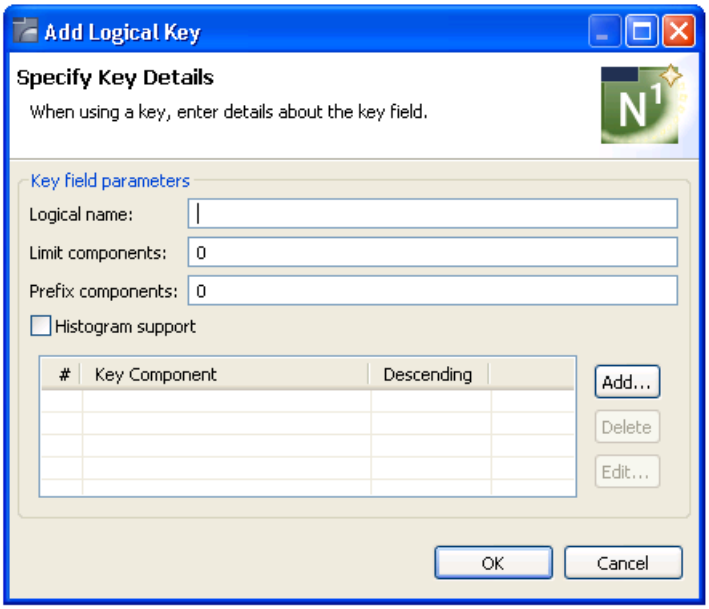
- Add a Logical Key
- Delete a Logical Key
- Edit a Logical Key

Add a Logical Key

To add a logical key

1 Select **Add** on the **Specify Additional Parameters** panel.

The **Specify Key Details** window is displayed. For example:



2 Define the following parameters for the additional key field:

Parameter	Description
Logical name	Name of the logical key for which you are defining details.
Limit components	Number of components of a superdescriptor (compound key) to use in the logical key. Use this option when the relational database table contains a superdescriptor with many components. To restrict the number of components, specify the limit in this field. For example, to use the first two components of the superdescriptor, enter "2". **Tip:** Using fewer components in the key may make accessing the key more efficient.
Prefix components	Prefix to use for components of a superdescriptor (compound key), which optimizes the generated SELECT statements when browsing by compound keys

Parameter	Description
	that have many components. You can use this option to define a browse object that requires specific values as the leading components. **Note:** When browsing Adabas or VSAM files by a single superdescriptor, efficiency is not affected by specifying prefix key components.
Histogram support	If this parameter is selected, the object-browse subprogram contains an additional histogram version of one or more logical key values. This allows the calling program to request a histogram be returned. Rather than returning all of the predefined columns for the object-browse subprogram, only the specific key column is returned along with a count of the number of records containing the key value. **Note:** This option is only allowed when the associated key has one key component.
Key Component	Up to five components for a logical key that maps to more than one component. For information, see *Specify Logical Key Components*.

3 Select **OK** to add the field.

Delete a Logical Key

To delete a logical key

1 Select the field you want to delete on the **Specify Additional Parameters** panel.

2 Select **Delete**.

The key is removed from the **Logical keys** table.

Edit a Logical Key

To edit a logical key

1 Select the field you want to edit on the **Specify Additional Parameters** panel.

2 Select **Edit**.

Or:

Double-click on the row in the **Key Component** section.

The **Specify Field Details** window is displayed, showing the current settings for the field.

3 Edit the field settings.

4 Select **OK** to save the changes.

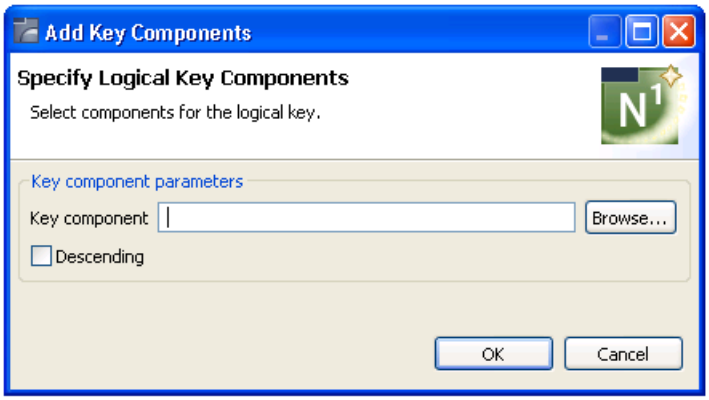
Specify Logical Key Components

Optionally, you can add up to five components for a logical key that maps to more than one component.

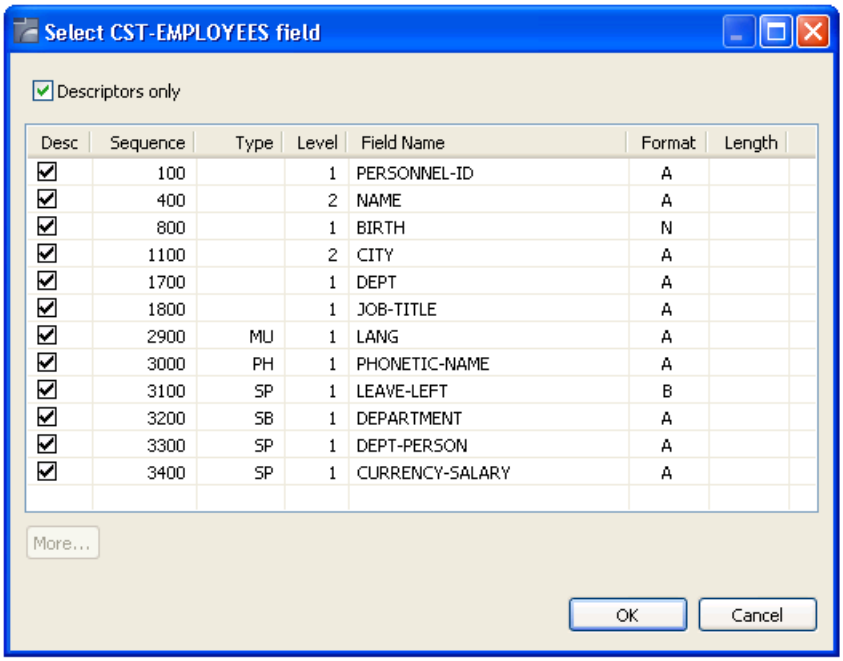
To add a logical key component

1 Select **Add** on the **Specify Additional Parameters** panel.

The **Specify Logical Key Components** window is displayed. For example:



2 Define the following parameters for the key component:

Parameter	Description
Key component	Type the name of the key component in **Key component** or select **Browse** to display a list of components for selection. For example:  Desc Sequence Type Level Field Name Format Length 100 1 PERSONNEL-ID A 400 2 NAME A 800 1 BIRTH N 1100 2 CITY A 1700 1 DEPT A 1800 1 JOB-TITLE A 2900 MU 1 LANG A 3000 PH 1 PHONETIC-NAME A 3100 SP 1 LEAVE-LEFT B 3200 SB 1 DEPARTMENT A 3300 SP 1 DEPT-PERSON A 3400 SP 1 CURRENCY-SALARY A You can specify either one superdescriptor or multiple individual descriptors. **Note:** To display all fields, deselect **Descriptors only**.
Descending	Indicates whether key component values are listed in ascending or descending sequence in the generated subprogram. To have the key component values listed in descending sequence, select this parameter. Otherwise, values are sorted in ascending sequence. **Note:** For Adabas and VSAM files, all components of a logical key must use the same sort order.

3 Select **OK** to save the settings.

Object-Maint-Dialog Wizard

This section describes the specification parameters for the Object-Maint-Dialog wizard. This wizard generates a character-based user interface (Natural program) for an object-maintenance process. The dialog component communicates with the user and invokes methods (data actions) implemented by the object-maintenance subprogram. To generate a complete maintenance process using Natural Construct's object-oriented approach, use this wizard in conjunction with the *Object-Maint-Subp* or *Object-Maint-Enhanced-Subp* wizard (which also generates the object PDA and restricted PDA).

This section covers the following topics:

- Specify Standard Parameters
- Specify Additional Parameters
- Specify Input Maps for Horizontal Panels
- Define Horizontal Panel Details
- Define Scroll Region Details

Note: For more information, refer to *Object-Maint-Dialog Model, Natural Construct Object Models*.

Specify Standard Parameters

The **Specify Standard Parameters** panel is displayed when the wizard is invoked; it is similar for all Object-Maint wizards.

To specify standard parameters

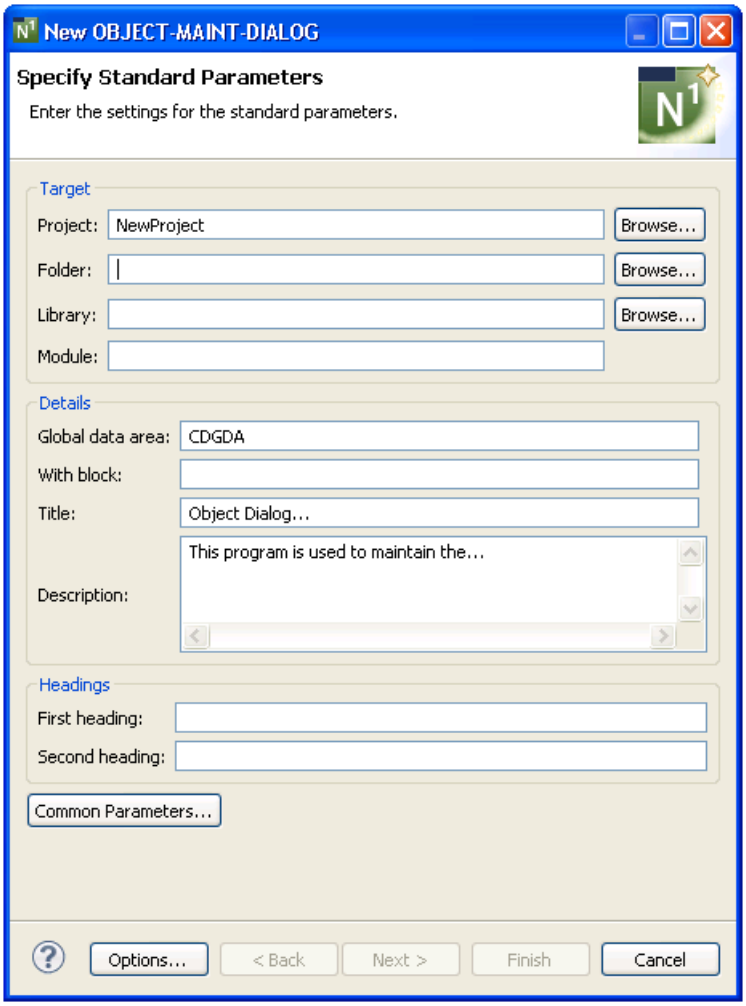
1 Open the context menu in the **Project Explorer** for the NaturalONE project in which you want to generate the modules.

   Or:

   Open the context menu in the **Project Explorer** for the library in which you want to generate the modules.

2 Select **Code Generation > New Using Construct Model > Object-Maint-Dialog**.

   The **Specify Standard Parameters** panel is displayed. For example:

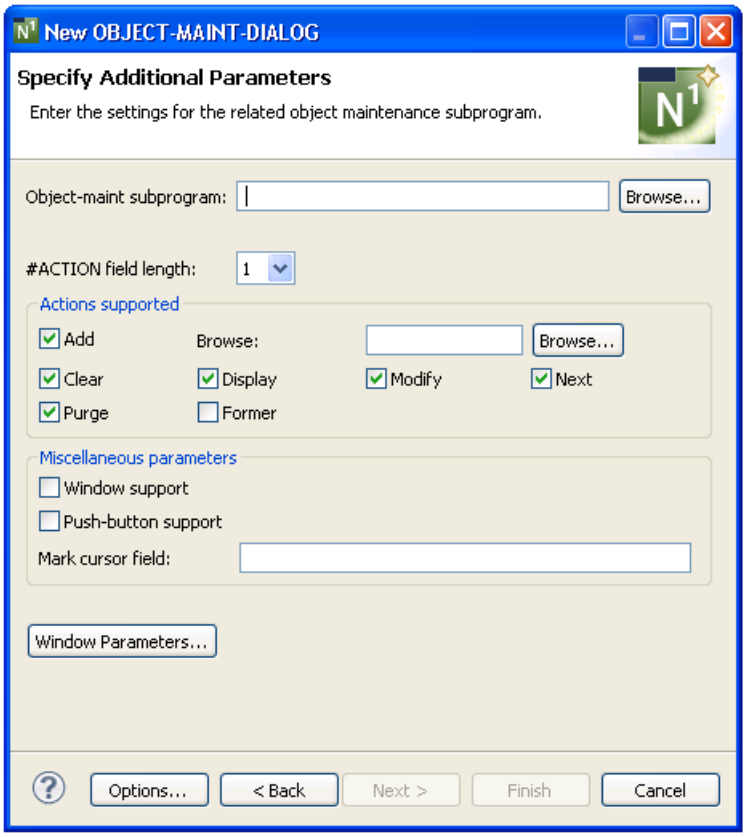


Many of the parameters on this panel are common to most wizards. For information, see *Specify Standard Parameters*. Optionally, you can:

Task	Procedure
Define common parameters, such as support for direct command processing, message numbers, and password checking.	Select **Common Parameters**. For information, see *Specify Common Parameters*.

3 Select **Next**.

The **Specify Additional Parameters** panel is displayed. For example:



Use this panel to define parameters for the related object-maint subprogram and, optionally, to define screen parameters.

Specify Additional Parameters

To specify additional parameters

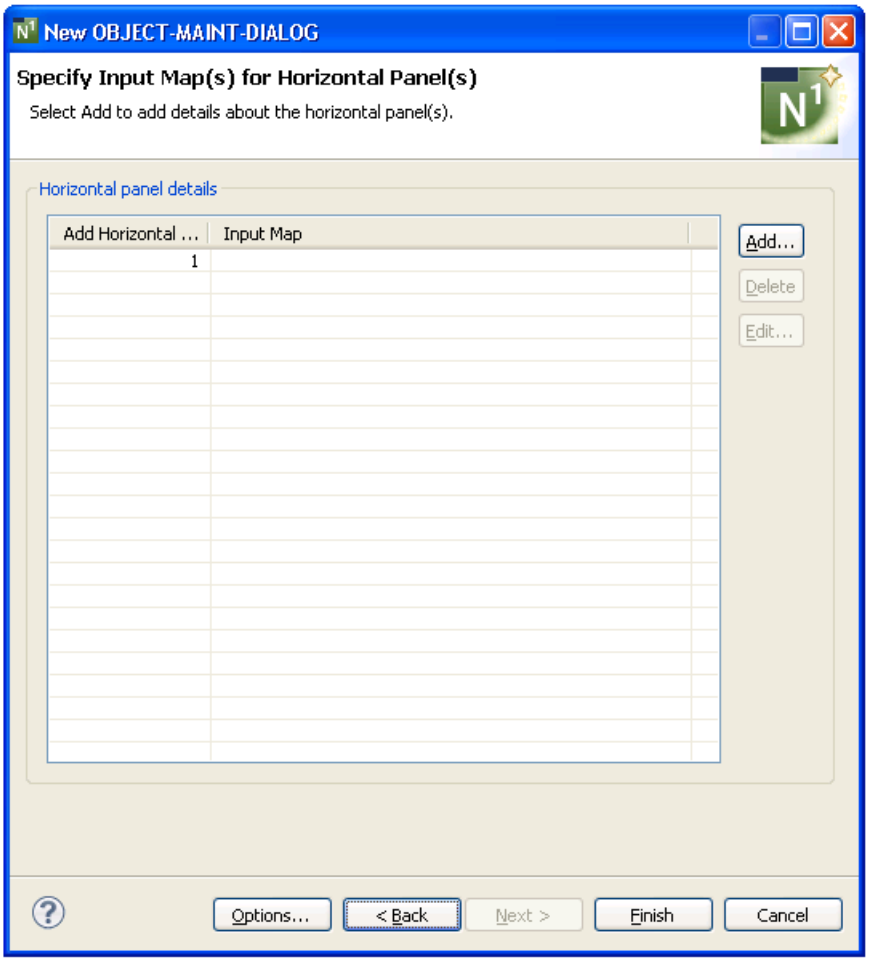
1 Define the following parameters:

Field	Description
Object maint subprogram	Name of the subprogram invoked by the generated dialog program. The specified subprogram must exist in the current library. Name of the object-maintenance subprogram used by the generated dialog program. Either type the name or select **Browse** to display the available subprograms for selection.
#ACTION field length	Length of the action field. By default, the length is "1" and all action fields except Former are marked. If you do not want the generated dialog program to perform a particular action, deselect the corresponding action field. At least one action must be selected. The available actions are:

Field	Description
	■ Add (adds the specified object) ■ Browse (name of the generated subprogram that supports the Browse action; either type the name or select **Browse** to display the available subprograms for selection) ■ Clear (clears the specified field values from the panel) ■ Display (displays the specified object) ■ Modify (modifies the specified object) ■ Next (displays the contents of the record having the next higher primary key value from the current key value) ■ Purge (removes the specified object) ■ Former (displays the contents of the record having the next lower primary key value from the current key value) **Note:** 1. To add user-defined actions, see *Add an Action*, *Natural Construct Generation*. 2. When generating an object-maintenance dialog program, this feature works together with two user exits. For information about these exits, see *SELECT-ADDITIONAL-ACTIONS* and *ADD-ACTION-PROCESSING*, *Natural Construct Generation*.
Window support	Indicates whether the output from the generated object-maintenance dialog program is displayed in a window instead of on a panel.
Push button support	Indicates whether actions can be selected by cursor.
Mark cursor field	Name of the field on the map where the cursor is automatically placed by the generated dialog program.
Window Parameters	Change the default window parameters. For information, see ***Change the Window Settings***.

2 Select **Next**.

The **Specify Input Map(s) for Horizontal Panel(s)** panel is displayed. For example:



Use this panel to define horizontal panels and layout maps and, optionally, add scroll regions for the horizontal panels.

Specify Input Maps for Horizontal Panels

By default, the generated dialog program uses one panel and you must specify a layout map for that panel.

To specify the layout map for panel 1

1 Select the "1" row in **Add Horizontal**.

The **Delete** and **Edit** buttons are enabled.

2 Select **Edit**.

The **Define Horizontal Panel Details** window is displayed. For information, see *Define Horizontal Panel Details*.

Note: You can also use the **Define Horizontal Panel Details** window to add additional horizontal panels. For information, see *Add a Horizontal Panel*.

3 Either type the name of the layout map in **Map** or select **Browse** to display a list of available maps for selection.

The **Specify Input Maps for Horizontal Panels** panel is redisplayed, showing the name of the layout map in **Input Map** for panel 1.

4 Select **Finish**.

The modules are generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

5 Save the generated modules.

At this point, you can:

- Define user exits for the dialog program. For information, see *Defining User Exits*.
- Use NaturalONE functionality to upload all generated modules to the server.

Define Horizontal Panel Details

Optionally, you specify up to nine horizontal panels for the generated dialog (you must specify a minimum of one panel). If more than one panel is specified, the left and right PF-keys are activated in the generated program to allow left and right scrolling between panels.

This section covers the following topics:

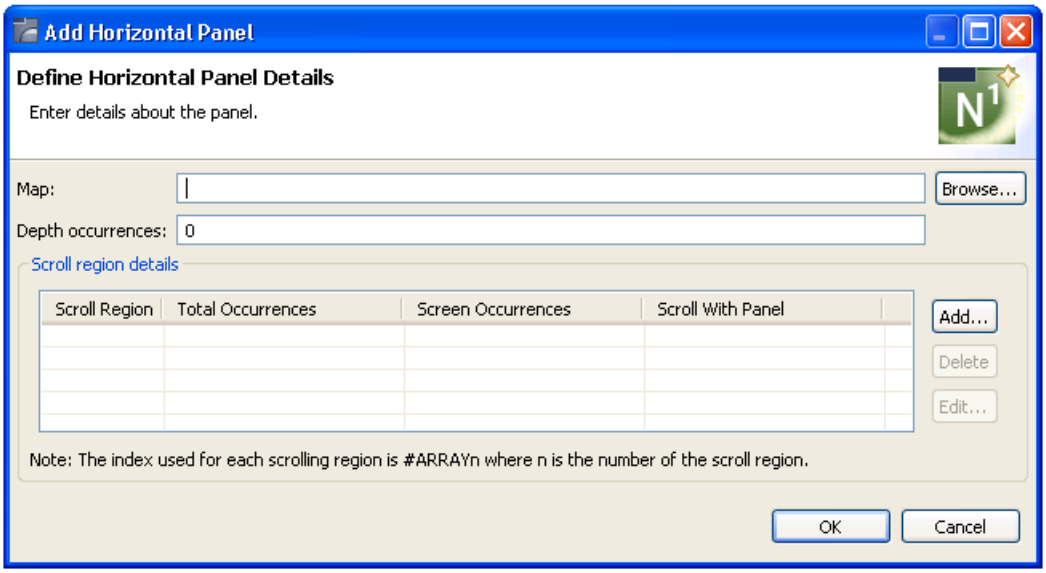
- Add a Horizontal Panel
- Delete a Horizontal Panel
- Edit a Horizontal Panel

Add a Horizontal Panel

To add a horizontal panel

1 Select **Add** on the **Specify Input Maps for Horizontal Panels** panel.

Define Horizontal Panel Details window is displayed. For example:



2 Define the following parameters for the horizontal panel:

Field	Description
Map	Name of the layout map used for the corresponding panel. Either type the name of the layout map or select **Browse** to display a list of available maps for selection.
Depth occurrences	To create a scroll region with a third dimension, specify the maximum depth occurrences value. For example, for a calendar with the months and days forming the first two dimensions (horizontal and vertical) and the year forming the third dimension (depth), you can specify "3" to scroll up to three yearly tables of calendar months and days, and within each yearly table, scroll vertically through the days. The Depth occurrences value applies to the 3rd dimension on a panel, which means that it applies to all 3-dimension arrays on the map when using the #DEPTH index variable. To allow the value of the #DEPTH variable to be changed, you can either place the #NEXT-DEPTH (P3) variable on the specified map or use PF-keys that you process in the AFTER-INPUT user exit.
Scroll region details	Information for up to four vertical scroll regions for each horizontal panel. To define a scroll region, select **Add**. For information, see *Define Scroll Region Details*.

3 Select **OK** to add the panel.

Delete a Horizontal Panel

To delete a horizontal panel

1 Select the panel you want to delete on the **Specify Input Maps for Horizontal Panels** panel.
2 Select **Delete**.

   The panel is removed from the **Scroll region details** table.

Edit a Horizontal Panel

To edit a horizontal panel

1 Select the panel you want to edit on the **Specify Input Maps for Horizontal Panels** panel.
2 Select **Edit**.

   Or:

   Double-click on the row in the **Scroll region details** table.

   The **Define Horizontal Panel Details** window is displayed, showing the current settings for the panel.
3 Edit the panel settings.
4 Select **OK** to save the changes.

Define Scroll Region Details

Optionally, you can define up to four vertical scroll regions (consisting of vertical arrays) for each horizontal panel. Scroll regions are only required for array fields. For example, assume you have an array field called #YEAR and each occurrence contains one month, but there is only enough space on the screen to display three months. In this case, a scroll region with a screen occurrences value of "3" is required.

In addition to scrolling through the months, you may also want to display the revenue for each month and have the revenue data change when the month changes. To do this, the starting index for #YEAR and #REVENUE must be the same. For example, assume the following:

```
#YEAR (A4/12)
#REVENUE (N5.1/12)
```

On the specified layout map, there should be three occurrences of both #YEAR and #REVENUE. For example, if these fields are in the first scroll region, use #ARRAY1 as the starting index for both fields. The object-maint dialog program will generate the appropriate code based on the values defined for the scroll region (for example, top left and bottom right). If the cursor is in the

area defined by the upper left and bottom right coordinates, then #ARRAY1 will be incremented appropriately when the forward or backward PF-keys are selected.

Tip: You can think of a two-dimensional (2D) array as a collection of many one-dimensional (1D) arrays. And you can think of a fixed instance of a third dimension of a three-dimensional (3D) array as a 2D array. Therefore, a vertical scroll region on a generated panel can consist of 1D, 2D, or 3D arrays.

This section covers the following topics:

- Add a Scroll Region
- Delete a Scroll Region
- Edit a Scroll Region

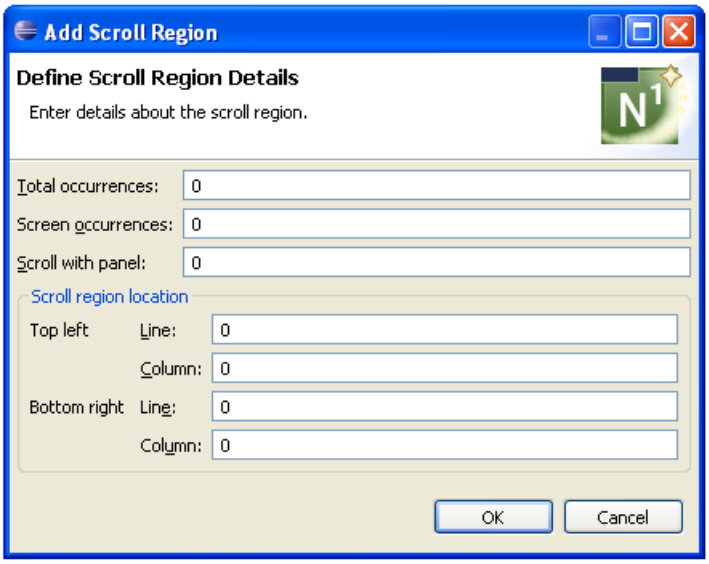
Add a Scroll Region

Note: If you add scroll regions, the specified map should contain array fields that match the specified values.

To add a scroll region

1 Select **Add** on the **Define Horizontal Panel Details** panel.

Define Scroll Region Details window is displayed. For example:



2 Define the following parameters for the scroll region:

Field	Description
Total occurrences	Total number of scrollable lines required for the scroll region. This value applies when the generated dialog program includes a line scroll feature to scroll: ■ Records in a secondary or tertiary file ■ Multiple-valued fields (MUs) ■ Periodic groups (PEs) The generated program ensures that the values assigned to the array index values (#ARRAY1 through #ARRAY4) do not exceed the total occurrences value for each array.
Screen occurrences	Total number of lines displayed on the panel at one time (used when the Total occurrences value is specified).
Scroll with panel	Panel number to force a particular *starting from* value for a scroll region on a panel (so it has the same value as another panel). Each panel maintains its own current values for #ARRAY*n*, where *n* is 1, 2, 3, or 4.
Scroll region location Top left	Location of the top left corner of the scroll region. A scroll region is always rectangular and is defined by specifying the panel coordinates of the top left and bottom right corners. Users can press the bkwrd and frwrd PF-keys to position the scroll regions backward and forward. The values for the top left corner are: ■ Line Starting line number (vertical axis). ■ Column Starting column number (horizontal axis).
Scroll region location Bottom right	Location of the bottom right corner of the scroll region. These values are: ■ Line Ending line number (vertical axis). ■ Column Ending column number (horizontal axis).

3 Select **OK** to add the scroll region.

Delete a Scroll Region

To delete a scroll region

1 Select the scroll region you want to delete on the **Define Horizontal Panel Details** panel.

2 Select **Delete**.

The scroll region is removed from the **Scroll region details** table.

Edit a Scroll Region

To edit a scroll region

1 Select the scroll region you want to edit on the **Define Horizontal Panel Details** panel.

2 Select **Edit**.

Or:

Double-click on the row in the **Scroll region details** table.

The **Define Scroll Region Details** window is displayed, showing the current settings for the panel.

3 Edit the scroll region settings.

4 Select **OK** to save the changes.

Object-Maint-Enhanced-Subp Wizard

This section describes the Object-Maint-Enhanced-Subp wizard, which generates an object-maintenance subprogram and corresponding PDAs. The generated subprogram updates all entities within an object and contains a full range of integrity checks (as defined by Predict relationships) and object semantics (in the form of Predict automatic rules or object manipulation within user exits). This wizard is similar to the Object-Maint-Subp wizard. The main difference between these wizards is that the Object-Maint-Enhanced-Subp wizard can generate large fields in the object PDA as dynamic fields. This allows long fields to occupy only the space required to pass the data to the database view. For example, one customer may require 1000 characters for delivery instructions and another customer only requires 50 characters. In the first case, 1000 characters will be placed in the parameter data area (PDA) and in the second case only 50 characters will be placed in the PDA.

The Object-Maint-Enhanced-Subp wizard allows you to take advantage of larger field sizes available in Natural and in the databases. In the past, an alphanumeric field in Natural was restricted to a length of 253 characters. To accommodate larger fields, you had to create an array of strings with a length of less than 254 characters each. This meant that words in a note, for example, may have been split across strings. Using this wizard, you can specify larger string sizes in the files

and in Natural to allow the entire note to fit in one string. The wizard can also generate code to truncate trailing blanks, which can needlessly increase the amount of data going into the PDA, and generate an error message when a user enters data into a field that is longer than what the database expects.

This section covers the following topics:

- Specify Standard Parameters
- Specify Additional Parameters
- Specify Input Parameters
- PROCESS-TRUNCATION-ROUTINE User Exit

Notes:

1. For more information about creating an object-maintenance process, refer to *Design Methodology*, *Natural Construct Generation*.
2. For information about the standard user exits, refer to *User Exits for the Generation Models*, *Natural Construct Generation*.
3. For information about the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.

Specify Standard Parameters

The **Specify Standard Parameters** panel is displayed when the wizard is invoked; it is similar for all Object-Maint wizards.

Note: To access this wizard, the specified project must be mapped to a version 8.2 or higher server environment.

To specify standard parameters

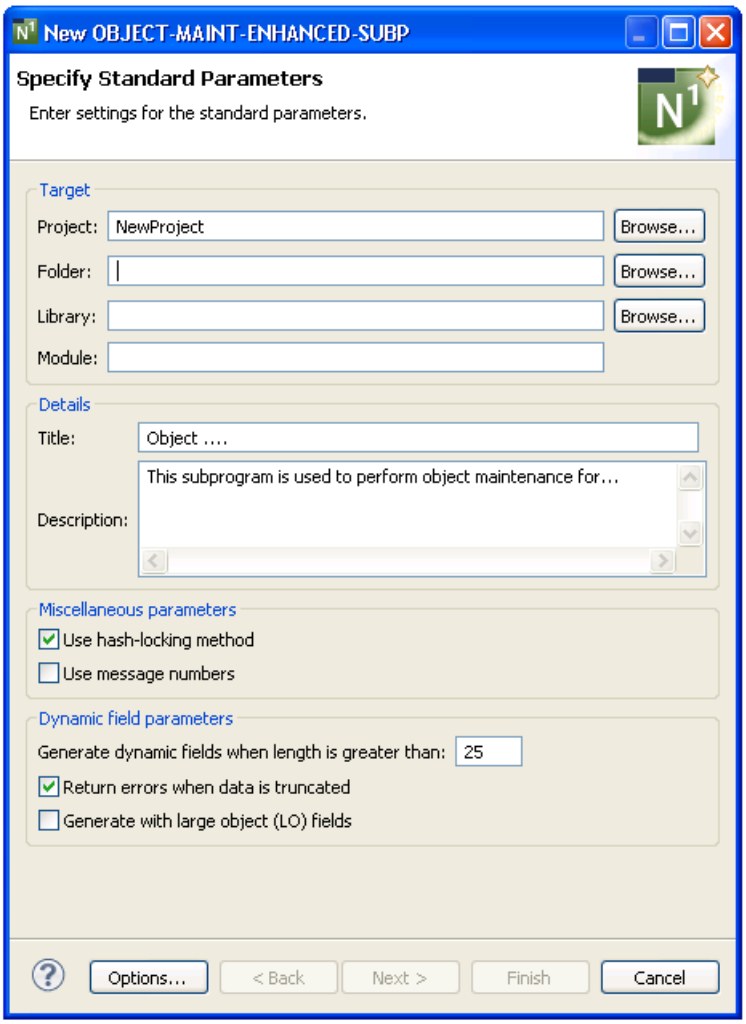
1 Open the context menu in the **Project Explorer** for the NaturalONE project in which you want to generate the modules.

Or:

Open the context menu in the **Project Explorer** for the library in which you want to generate the modules.

2 Select **Code Generation > New Using Construct Model > Object-Maint-Enhanced-Subp**.

The **Specify Standard Parameters** panel is displayed. For example:



Many of the parameters on this panel are common to most wizards. For information, see *Specify Standard Parameters*. The **Miscellaneous parameters** are identical to those for the Object-Maint-Subp wizard. For information about these fields, see *Object-Maint-Subp Wizard*.

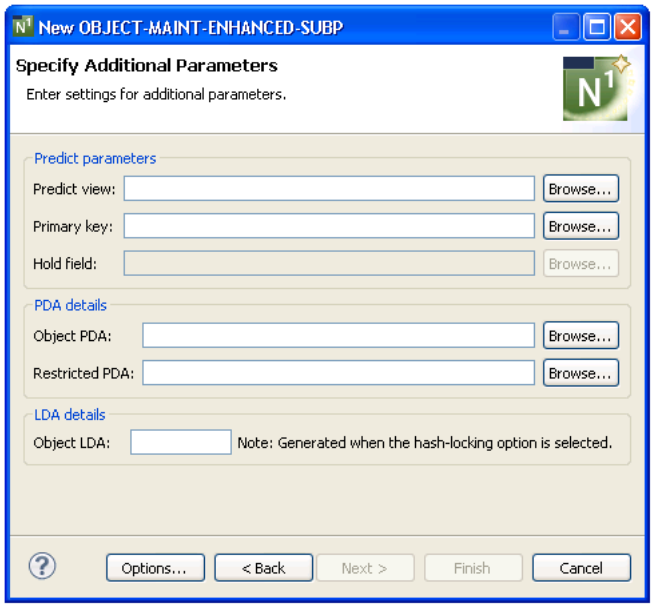
Optionally, you can:

Task	Procedure
Generate dynamic fields into the object PDA when the size of the source field is larger than the number of characters specified in the **Generate dynamic fields when length is greater than** *nnn* field.	Select **Generate dynamic fields when length is greater than** *nnn*, where *nnn* is a number less than 1000 and 0 indicates the data PDA contains the same lengths as the DDM. For example, if the specified cutoff length is 50 and a field is defined in the DDM as alphanumeric 100, an (A) DYNAMIC field will be generated into the object PDA instead of an (A100). **Note:**

Task	Procedure
	1. If the cutoff length is "0", the field sizes in the PDA will be the same as those in the DDM. 2. If a field is affected by the cutoff length, it may not be part of a redefined field. 3. If the cutoff length is a negative number, the length is converted to a positive value (for example, "-10" is converted to "10").
Return an error message when the data in a dynamic field is larger than its source database field and the data will be truncated by the subprogram (i.e., dynamic fields in the PDA but not in the file view).	Select **Return errors when data is truncated** to have the generated maintenance subprogram return an error message when data is truncated. For example, you may have a text field with a variable length for descriptive information that you want to set to 1000 characters. Since a dynamic field can handle more than 1000 characters, you must decide what happens if the user enters more. One option is to let the user enter whatever they want and the subprogram will truncate any data over the limit when it stores it in the database. Another option is to generate error messages when the user exceeds the limit and/or to stop the processing. When the subprogram is generated with the truncation option, Construct will provide error messages and a user exit to define how to handle the error. Within this user exit, Construct generates a list of all affected fields (i.e., are dynamic fields in the PDA but not in the file view) and allows you to change the value for ##RETURN-CODE or add an ESCAPE ROUTINE to continue with processing when an error occurs. **Note:** Truncation errors and messages are processed in the PROCESS-TRUNCATION-ROUTINE user exit. For information, see *PROCESS-TRUNCATION-ROUTINE User Exit*.
Maintain large object (LO) fields with the generated maintenance subprogram.	Select **Generate with large object (LO) fields**.

3 Select **Next**.

The **Specify Additional Parameters** panel is displayed. For example:



Use this panel to define additional parameters for your object-maint-enhanced subprogram.

Specify Additional Parameters

The fields on this panel are identical to the fields on the **Specify Additional Parameters** panel for the Object-Maint-Subp wizard. The only difference is that the **Generate** options have been removed for the **Object PDA** and **Restricted PDA** fields. These parameter data areas will always be generated or regenerated with the Object-Maint-Enhanced-Subp wizard, since the field definitions may change when dynamic fields are processed.

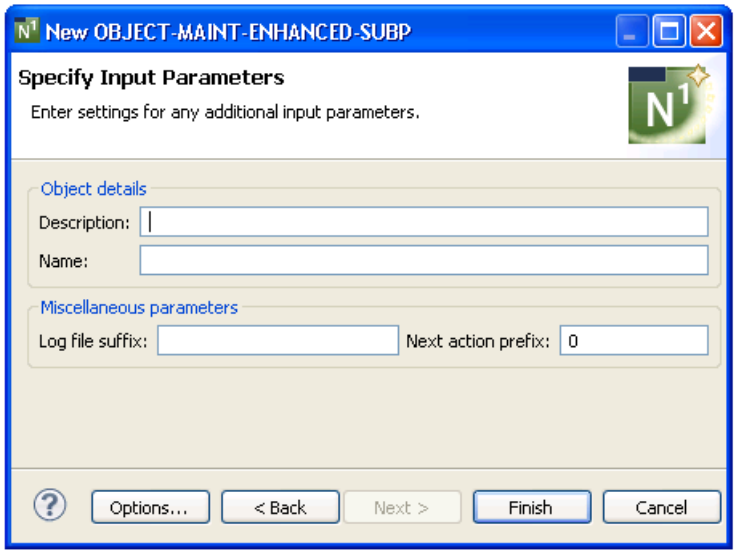
To define additional parameters

1 Specify the additional parameters for the object-maint-enhanced subprogram.

For more information, see *Specify Additional Parameters*.

2 Select **Next**.

The **Specify Input Parameters** panel is displayed. For example:



Use this panel to define additional input parameters for your object-maint-enhanced subprogram.

Specify Input Parameters

The fields on this panel are identical to the fields on the **Specify Input Parameters** panel for the Object-Maint-Subp wizard.

To define additional input parameters

1 Specify the additional input parameters for the object-maint-enhanced subprogram.

For more information, see *Specify Input Parameters*.

2 Select **Finish**.

The modules are generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

3 Save the generated modules.

At this point, you can:

- Use the NaturalONE Testing option to test the subprogram. For information, see *Test a Subprogram Directly* in *Application Testing*.
- Define user exits for the subprogram. For information, see *Defining User Exits*.
- Use NaturalONE functionality to upload all generated modules to the server.

PROCESS-TRUNCATION-ROUTINE User Exit

This user exit can be used in the generated subprogram to define truncation routines and error messages for dynamic fields. It is a Conditional exit and available when the PDA for the subprogram contains dynamic fields in the object PDA that represent fixed-length fields in the database.

When you select the PROCESS-TRUNCATION-ROUTINE user exit, the following code is generated into the exit:

```
Module ............. ModName                                                  ↩

Title .............. Object ....
>                                                  > + ABS: X X-Y: _ S    5 L    1
  All  ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
  0010 DEFINE EXIT PROCESS-TRUNCATION-ROUTINE
  0020 /* Start of PROCESS-TRUNCATION-ROUTINE user exit
  0030 /* note that the ##RETURN-CODE can be changed or
  0040 /* ESCAPE ROUTINE can be added so that one doesn't stop the program.
  0050 END-EXIT                                                                ↩
```

To allow processing to continue when a truncation error occurs, you can change the value for ##RETURN-CODE or add an ESCAPE ROUTINE.

Object-Maint-Subp Wizard

This section describes the Object-Maint-Subp wizard, which generates a subprogram that maintains complex data objects. The subprogram updates all entities within an object and contains a full range of integrity checks (as defined by Predict relationships) and object semantics (in the form of Predict automatic rules or object manipulation within user exits).

This section covers the following topics:

- Specify Standard Parameters
- Specify Additional Parameters
- Specify Input Parameters

Notes:

1. For information about the Object-Maint-Subp model, refer to *Object-Maint-Subp Model*, *Natural Construct Object Models*.
2. For more information about creating an object-maintenance process, refer to *Design Methodology*, *Natural Construct Generation*.

Specify Standard Parameters

The **Specify Standard Parameters** panel is displayed when the wizard is invoked; it is similar for all Object-Maint wizards.

To specify standard parameters

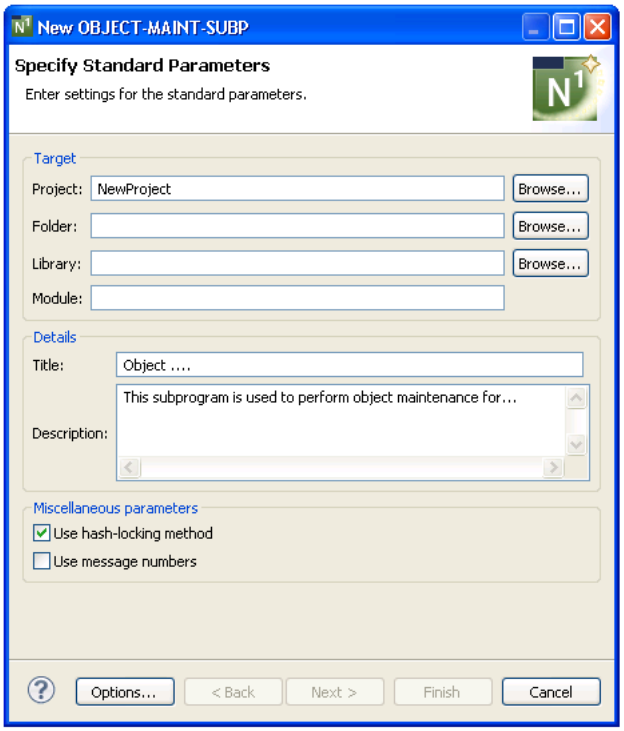
1 Open the context menu in the **Project Explorer** for the NaturalONE project in which you want to generate the modules.

Or:

Open the context menu in the **Project Explorer** for the library in which you want to generate the modules.

2 Select **Code Generation > New Using Construct Model > Object-Maint-Subp**.

The **Specify Standard Parameters** panel is displayed. For example:

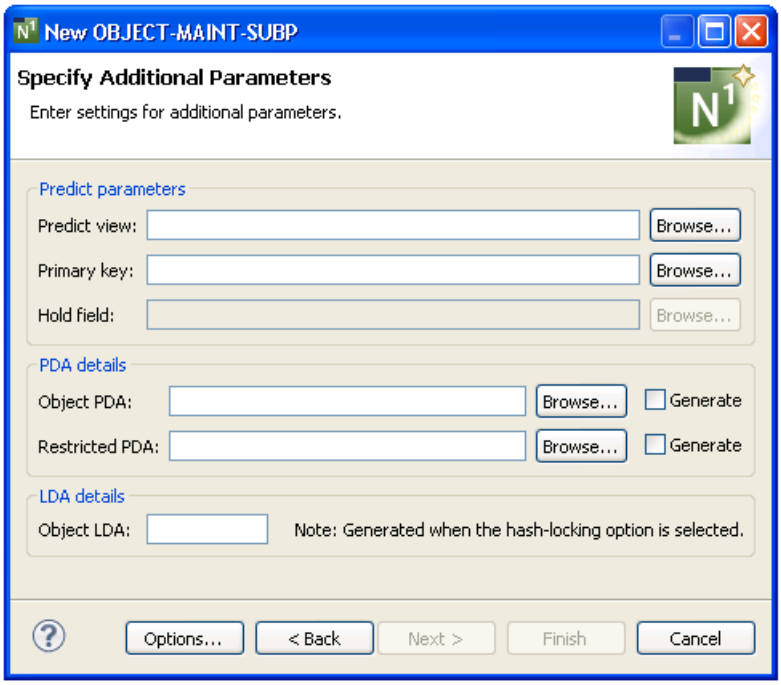


Many of the parameters on this panel are common to most wizards. For information, see *Specify Standard Parameters*. Optionally, you can:

Task	Procedure
Use the hash-locking method of optimistic record locking (instead of the timestamp method).	Select **Use hash-locking method**. When this method is selected, the **Hold field** parameter is disabled on the second specification panel and the **Object LDA** parameter is enabled. To enable the Hold field, deselect **Use hash-locking method**.
Use message numbers for all REINPUT and INPUT messages (instead of message text).	Select **Use message numbers**. When this option is selected, message numbers rather than message text will be used for all REINPUT and INPUT messages. **Note:** Use the same technique consistently throughout your application, since passing messages between modules using different techniques will not always produce the desired results.

3 Select **Next**.

The **Specify Additional Parameters** panel is displayed. For example:



Use this panel to define additional parameters for your object-maintenance subprogram.

Specify Additional Parameters

To specify additional parameters

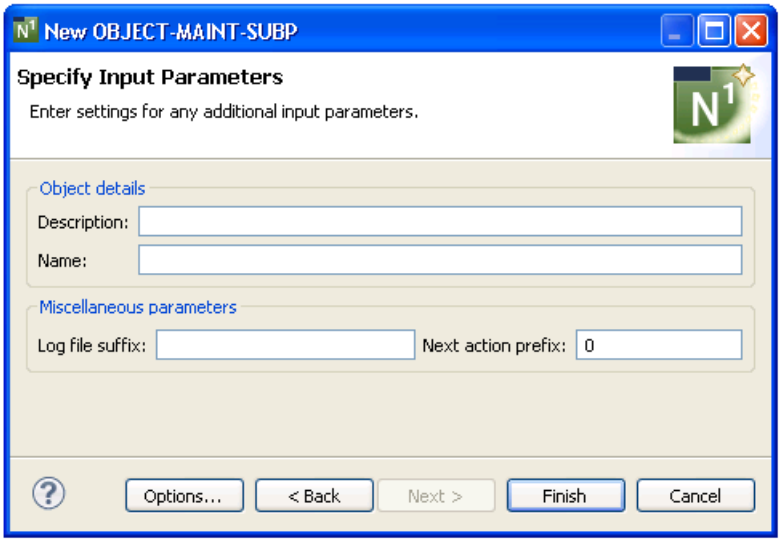
1 Define the following parameters:

Parameter	Description
Predict view	Name of the Predict view used by the generated subprogram. The view must be defined in Predict. Either type the name or select **Browse** to display the available views for selection.
Primary key	Name of the key in Predict for the primary file. This key becomes the primary key to access the view for maintenance. The key can be a descriptor, superdescriptor, or subdescriptor. If the key does not exist in the specified Predict file, an error message is displayed. **Note:** When Predict is used and the primary key is specified in the file, this parameter is not required.
Hold field	Name of the field used to logically protect the record against intervening update or delete actions. Because an object-maintenance subprogram does not use the record-holding facilities of the DBMS to lock records during a GET operation, a *hold* field must exist in the primary file for the object. Valid data types are: ■ T *TIMX ■ A10 *TIME ■ B8 *TIMESTMP ■ N7 *TIMN ■ A26 *TIMX (DB2 time stamp format) ■ If the format is none of the above, it must be numeric. **Note:** This field is enabled when the timestamp method for record locking is selected on the **Specify Standard Parameters** panel (i.e., the **Use hash-locking method** option is not selected).
Object PDA	Name of the object parameter data area (PDA) that defines the rows returned to the object-maint subprogram and the columns within each row. Either type the name or select **Browse** to display the available PDAs for selection. Alternatively, you can select **Generate** to have the data area generated by the wizard. The generated object PDA contains one column for each field defined in the specified Predict view (as well as additional columns). You can remove any fields that are not components of the primary key. **Note:** When creating a new specification, this field is filled in by default with the first five bytes of the subprogram name, plus the suffix "ROW".
Restricted PDA	Name of the restricted PDA that stores data, such as the last sort key, the last starting value, the last row returned, etc. so that the next set of consecutive records is returned

Parameter	Description
	to the caller. Either type the name or select **Browse** to display the available PDAs for selection. Alternatively, you can select **Generate** to have the data area generated by the wizard. **Note:** 1. The contents of this data area should not be altered by the calling module. 2. When creating a new specification, this field is filled in by default with the first five bytes of the subprogram name, plus the suffix "PRI".
Generate	Indicates whether an existing object PDA or restricted PDA is regenerated. Regeneration is required when fields have changed in the file. If the PDAs do not exist, they will be automatically generated by the wizard.
Object LDA	Name of the object local data area generated for the object-maintenance subprogram. **Note:** This field is enabled when the **Use hash-locking method** option for record locking is selected on the **Specify Standard Parameters** panel.

2 Select **Next**.

The **Specify Input Parameters** panel is displayed. For example:



Use this panel to define additional input parameters for your object-maintenance subprogram.

Specify Input Parameters

To specify input parameters

1 Define the following parameters:

Parameter	Description
Description	Object description used in messages. If you specify "Person", for example, messages are displayed as "Person not found" and "Person displayed".
Name	Name of the level 1 structure used to qualify the fields in the object PDA. (It is easier to identify the source of these attributes when the PDA name is used for this purpose.) The object name should be kept to a reasonable length. **Note:** The object name cannot match the name of a file included in the object, nor any field in the object.
Log file suffix	If you want to log objects, you have to create a log file corresponding to each entity within the object. The name of the log file is the name of the object file concatenated with the suffix specified here. For example, if the object consists of the NCST-ORDER-HEADER and NCST-ORDER-LINES entities and you specify "-LOG", the log file names are NCST-ORDER-HEADER-LOG and NCST-ORDER-LINES-LOG. The following fields are required in the log file that corresponds to the header entity in the object: ■ LOG-TIME Assigned with *TIMX for T format or *TIMN for N7 format. ■ LOG-DATE Assigned with *DATX for D format or *DATN for N8 format. (If LOG-TIME has an embedded date, such as *TIMX, this field is not required.) ■ LOG-TID Assigned with *INIT-ID. ■ LOG-USER Assigned with *INIT-USER. ■ LOG-ACTION Assigned with the #ADD, #MODIFY, or #PURGE log action codes, which are defined in the CDACTLOG local data area. You can initialize the values for these log action codes within CDACTLOG to suit your environment. In the log files corresponding to the sub-entities in the object, only the LOG-ACTION field is required.

Parameter	Description
	Note: For relational databases, use the underscore (_) character instead of the dash (-) for the log field names (LOG_TIME, LOG_DATE, LOG_TID, LOG_USER, LOG_ACTION).
Next action prefix	If the primary key is compound or redefined into various components, supply a value to limit the number of prefixed components confined on the Next action. This allows the subprogram to maintain objects with a common prefix value. For example, if the primary key is made up of Company + Account + Division and you do not want the Next action to span the Division values, specify "2". Specify "1" if the Next action is to be limited to the current Company value.

2 Select **Finish**.

The modules are generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

3 Save the generated modules.

At this point, you can:

- Use the NaturalONE Testing option to test the subprogram. For information, see *Test a Subprogram Directly* in *Application Testing*.
- Define user exits for the subprogram. For information, see ***Defining User Exits***.
- Use NaturalONE functionality to upload all generated modules to the server.

Quit Wizard

This section describes the specification parameters for the Quit wizard. This wizard generates a quit program that releases resources used by an application. It displays a confirmation window that overlays the host panel and gives users the option of quitting an application entirely or resuming where they left off. The name of the quit program is assigned to the DIALOG-INFO.##QUIT global variable in a Natural Construct-generated startup program.

Specify Standard Parameters

This section describes the **Specify Standard Parameters** panel for the Quit wizard. This panel is the only specification panel for the wizard.

To specify standard parameters

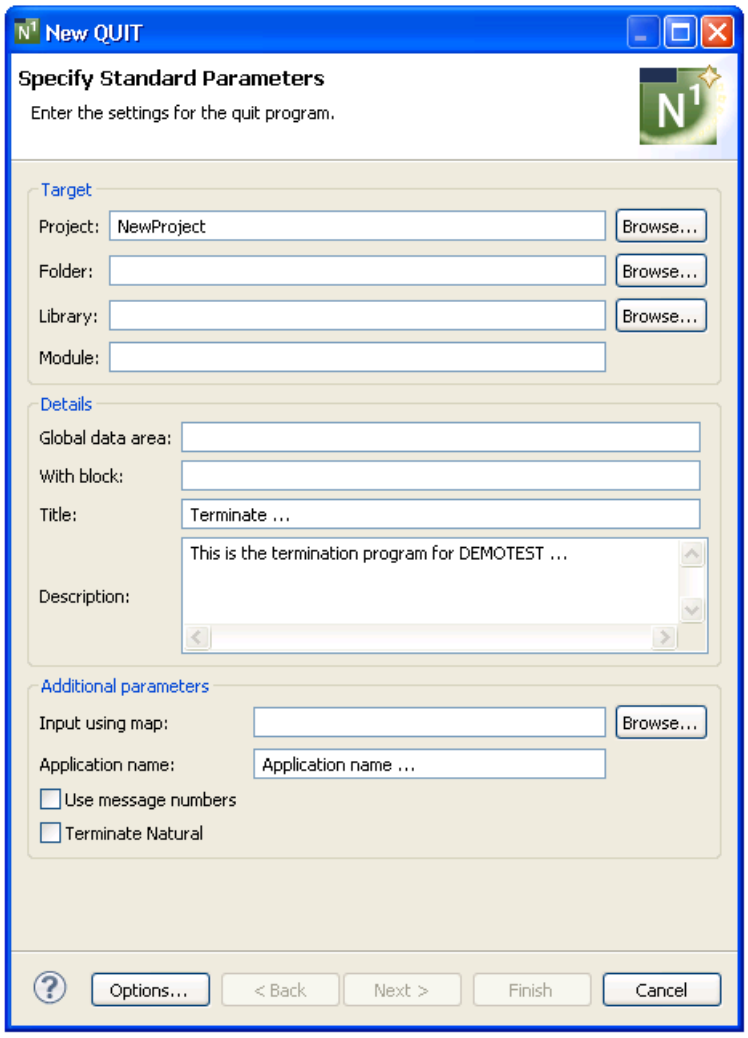
1 Open the context menu in the **Project Explorer** for the NaturalONE project in which you want to generate the modules.

Or:

Open the context menu in the **Project Explorer** for the library in which you want to generate the modules.

2 Select **Code Generation > New Using Construct Model > Quit**.

The **Specify Standard Parameters** panel is displayed. For example:



Many of the parameters on this panel are common to most wizards. For information, see *Specify Standard Parameters*. Optionally, you can use this panel to:

Task	Procedure
Provide the name of the layout map used for the generated quit program.	Type the name of the map in **Input using map** or select **Browse** to display the available maps for selection.
Provide a name for the application that will be used in confirmation messages.	Type the name in **Application name**.
Use message numbers for all REINPUT and INPUT messages.	Select **Use message numbers**. When this option is selected, message numbers rather than message text will be used for all REINPUT and INPUT messages. **Note:** Use the same technique consistently throughout your application, since passing messages between modules using different techniques will not always produce the desired results.
Have the quit program issue a Natural terminate command.	Select **Terminate Natural**. By default, the generated quit program: ■ Restores the default Natural error trapping ■ Sets the window size to the physical panel size ■ Releases the Natural stack ■ Backs out all outstanding database updates ■ Issues a Natural STOP command

3 Select **Finish**.

The quit program is generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

4 Save the generated module.

At this point, you can:

- Define user exits for the quit program. For information, see *Defining User Exits*.
- Use NaturalONE functionality to upload all generated modules to the server.

Startup Wizard

This section describes the specification parameters for the Startup wizard, which generates a startup program for an application. These programs (often named Menu) initialize global variables and invoke the main menu program.

This section covers the following topics:

- Specify Standard Parameters

Specify Standard Parameters

This section describes the **Specify Standard Parameters** panel for the Startup wizard. This panel is the only specification panel for the wizard.

To specify standard parameters

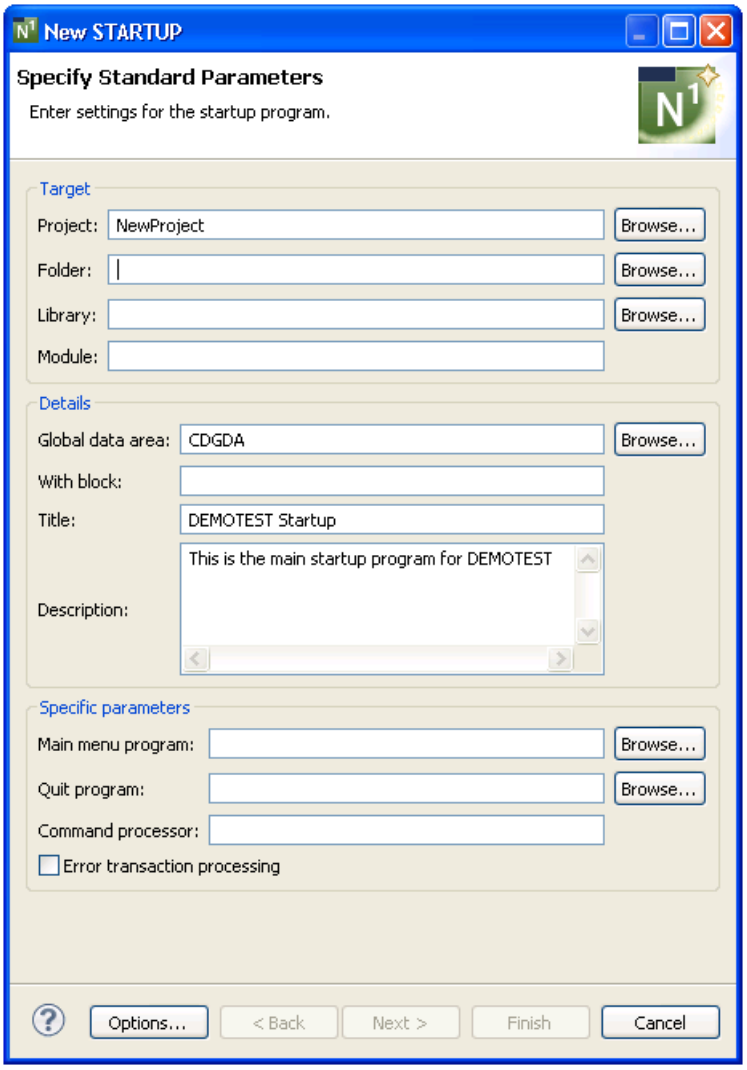
1 Open the context menu in the **Project Explorer** for the NaturalONE project in which you want to generate the modules.

   Or:

   Open the context menu in the **Project Explorer** for the library in which you want to generate the modules.

2 Select **Code Generation > New Using Construct Model > Startup**.

   The **Specify Standard Parameters** panel is displayed. For example:



Many of the parameters on this panel are common to most wizards. For information, see *Specify Standard Parameters*.

3 Define the following parameters:

Parameter	Description
Main menu program	Name of the program invoked by the startup program. This is usually the first panel displayed when a user issues the Natural MENU command.
Quit program	Name of the program invoked when a user ends a session. **Tip:** If no special cleanup is required when the program terminates, you can use the CD-QUIT program supplied with Natural Construct.

Optionally, you can:

Task	Procedure
Provide the name of a Natural command processor to process commands entered on the direct command line.	Type the name in **Command processor**. The specified command processor must have been created using the Natural SYSNCP systems utility.
Assign Natural Construct's default error transaction program (CDERRTA) to the *ERRORTA system variable.	Select **Error transaction processing**. For information about *ERRORTA, see the Natural documentation.

4 Select **Finish**.

The startup program is generated using the current specifications. When generation is complete, the available user exits are displayed in the **Outline** view.

5 Save the generated module.

At this point, you can:

- Define user exits for the startup program. For information, see *Defining User Exits*.
- Use NaturalONE functionality to upload all generated modules to the server.

Change the Dynamic Attribute Characters

This section describes the **Specify the Dynamic Attribute Parameters** window, which allows you to define up to four attributes, one of which must be the return to default display attribute (Default return field).

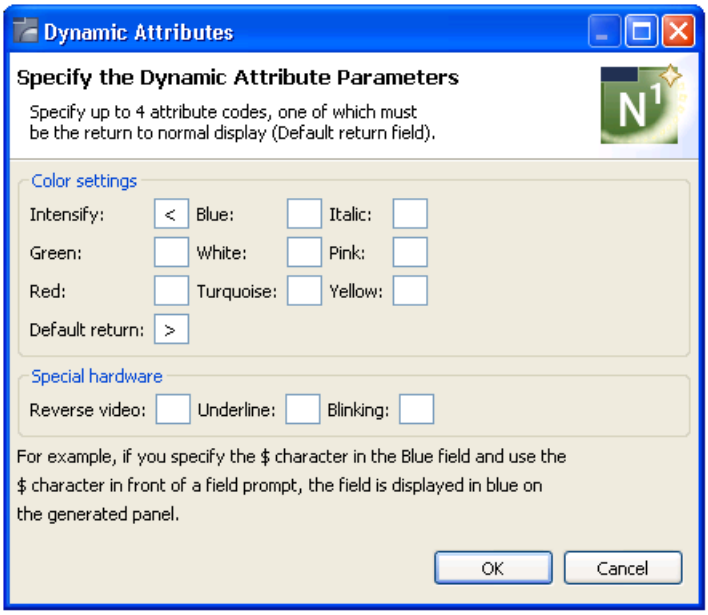
Notes:

1. To use some of the attributes listed in this window, special hardware is required.
2. For a description of the attributes and valid parameters for the fields, see the applicable Natural documentation.
3. Avoid using terminal control, alphabetic, and numeric characters when defining dynamic attributes.
4. If you are using Com-Plete, or cross-generating applications to run on a platform where Com-Plete is in use, also avoid using stacking characters.

To change the dynamic attribute characters

1 Select **Dynamic Attributes**.

The **Specify the Dynamic Attribute Parameters** window is displayed. For example:



2 Define up to four dynamic attributes, one of which must be the return to default display attribute (Default return field).

3 Select **OK** to save the settings.

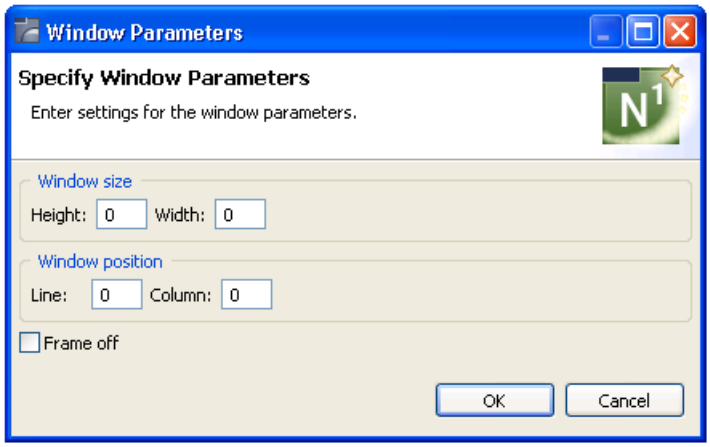
Change the Window Settings

This section describes the **Specify Window Parameters** window, which defines window parameters such as the height, width and position of the generated window, as well as whether it has a frame or not.

To change the window settings

1 Select **Window Parameters**.

The **Specify Window Parameters** window is displayed. For example:



2 Define the following parameters:

Parameter	Description
Height	Number of lines the window will span.
Width	Number of columns the window will span.
Line	Number of lines from the top of the panel to the top of the window.
Column	Number of columns from the left side of the panel to the left side of the window. The line and column values form the top left corner of the window.
Frame off	Determines whether the window is displayed with or without a border. Select this option to display the window without a border (frame).
Title	Title used for the window.

3 Select **OK** to save the settings.

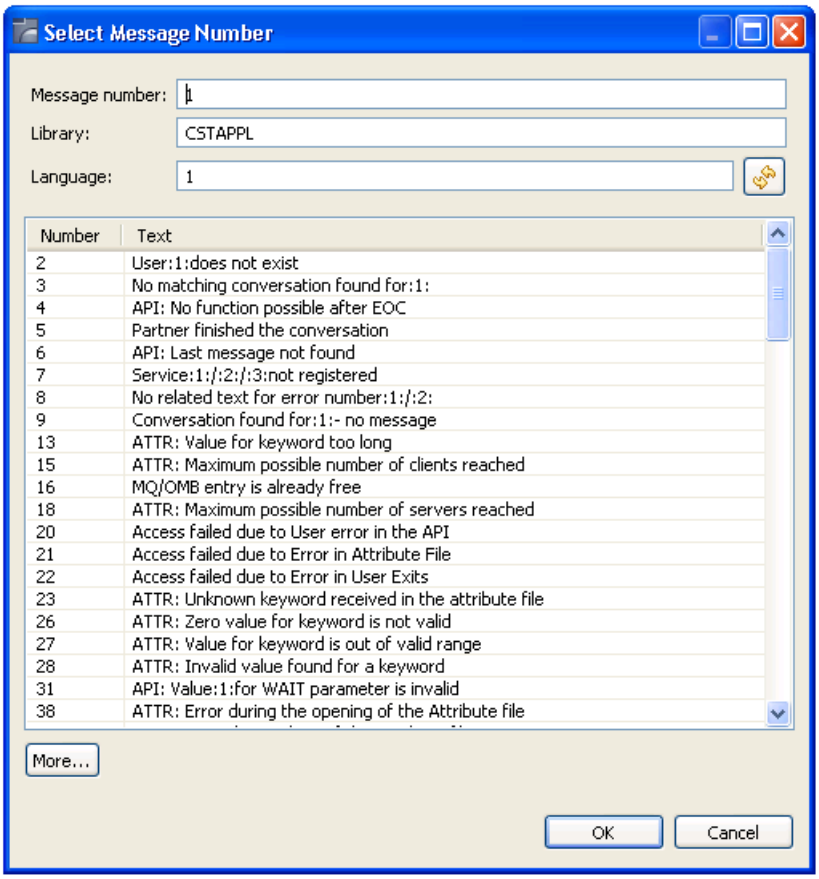
Select a Message Number

This section describes the **Select Message Number** window, which displays the available SYSERR numbers for selection.

To select a SYSERR number

1 Select **Browse** for **Prompt** on the **Specify Additional Parameters** panel.

The **Select Message Number** window is displayed. For example:



This window displays the available SYSERR numbers for selection.

2 Define the following parameters:

Parameter	Description
Message number	Number of the SYSERR listing in the specified library (by default, "1"). To display a different SYSERR listing, type a new number and select to display the SYSERR numbers in the specified library, beginning at the new number.
Library	Name of the library containing the SYSERR numbers (by default, "CSTAPPL"). To change the library, type a new library name and select to display the SYSERR numbers in the specified library.
Language	Code for the language number (by default, "1" for English). To change the language, type a new language code and select to display the SYSERR numbers in the specified language.
More	Displays additional SYSERR numbers (when more than 100 numbers are available). To display the next 100 SYSERR numbers, select **More**. Once all numbers have been displayed, the button is disabled.

3 Select the SYSERR number you want to use in the table.

4 Select **OK**.

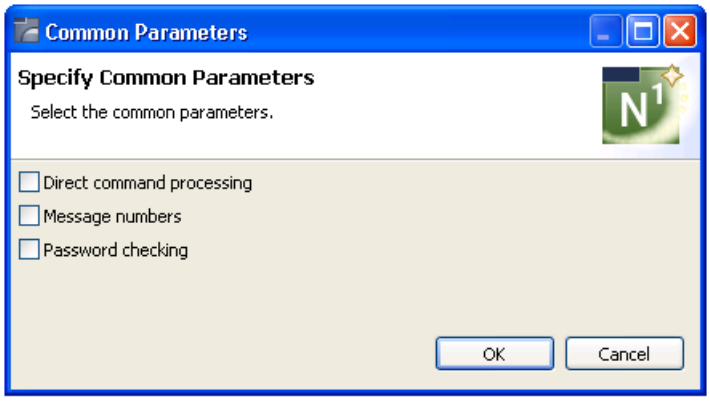
Specify Common Parameters

This section describes the **Specify Common Parameters** window, which defines common parameters such as support for direct command processing, message numbers, or password checking.

To define common parameters

1 Select **Common Parameters**.

The **Specify Common Parameters** window is displayed. For example:



2 Select one or more of the following options:

Option	Description
Direct command processing	Select this option to enable direct command processing.
Message numbers	Select this option to use message numbers rather than message text for all REINPUT and INPUT messages. **Note:** Use the same technique consistently throughout your application, since passing messages between modules using different techniques will not always produce the desired results.
Password checking	Select this option to enable password checking. To include password checking, you must also set up a password file. For information, see *Set Up a Password File*.

3 Select **OK** to save the settings.

Set Up a Password File

You can specify password checking for many of the generated modules. Natural Construct builds the mechanism for password checking into your modules by including the CCPASSW copycode member. Within this copycode, the CDPASSW subprogram is invoked and passed the module and library names.

To include password checking, you must set up a password file. The file is keyed on the module name used to catalog the module and the library name used to generate the module.

The password file can be a view of any file with Natural-Construct-Password as the data definition module (DDM) name. The view must contain the following fields:

Field	Format
PASSWORD-KEY	A40 (32-character library name, plus an 8-character module name)
PASSWORD	A8 (8-character password)

When a user attempts to invoke the module, the CDPASSW subprogram reads the password file. If the module/library name combination exists in the file and does not have a password, the user can invoke the module. If the module/library name combination exists and has a password, the user must enter the correct password before the module is invoked. If a user enters five incorrect passwords, execution is aborted.

If you specify password checking, you must modify the CDPASSW subprogram to include a valid password view and any final processing you want to perform and then catalog the modified subprogram. For more specific password checking, you can modify the CCPASSW copycode member (to call a different subprogram) or modify the CDPASSW subprogram (to refine your security standards).

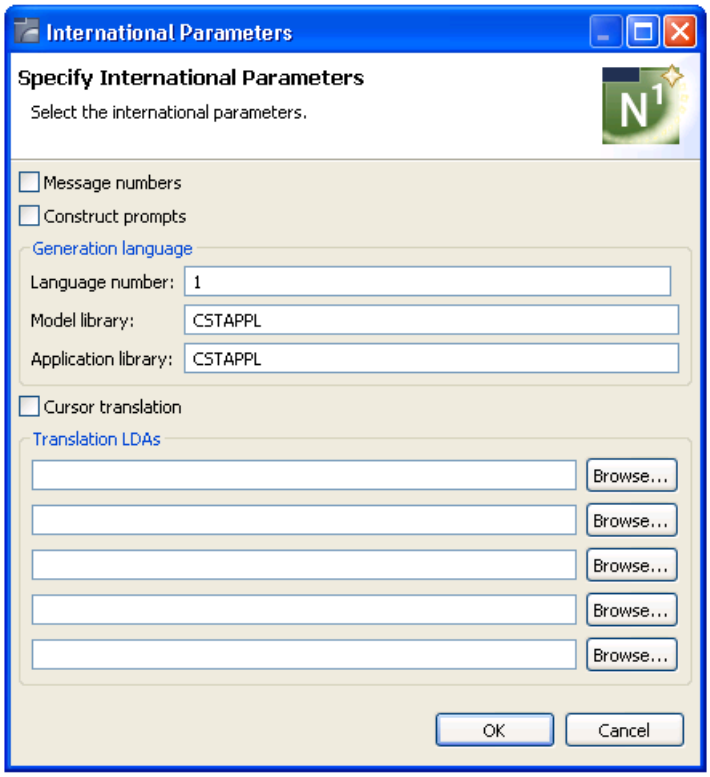
Specify International Parameters

This section describes the **Specify International Parameters** window, which defines the language used to display text on generated panels. You can define international parameters for modules generated using the Object-Browse-Dialog and user exit models.

To specify international parameters

1 Select **International Parameters**.

The **Specify International Parameters** window is displayed. For example:



2 Define the following parameters:

Parameter	Description
Message numbers	Type of messages used. When this option is selected, the generated code uses message numbers rather than message text.
Construct prompts	Type of prompts used. When this option is selected, the model generates Natural Construct-style prompts (for example, 1 of 2).
Language number	Code for the language used when generating message text. The default is 1 (English).
Model library	Name of the SYSERR message library used to retrieve common message text. The default is CSTAPPL.
Application library	Name of the SYSERR library used to retrieve message text for user-defined SYSERR references. This parameter is only applicable to modules generated using the Object-Browse-Dialog wizard. If you do not specify an application library, the Model library value is used.
Cursor translation	When this option is selected, the generated code supports cursor-sensitive translation (users can modify or translate panel text dynamically in translation mode).
Translation LDAs	Names of the translation local data areas (LDAs) used by generated modules. You can specify up to five translation LDAs. Either type the name or select **Browse** to display the available data areas for selection.

3 Select **OK** to save the settings.

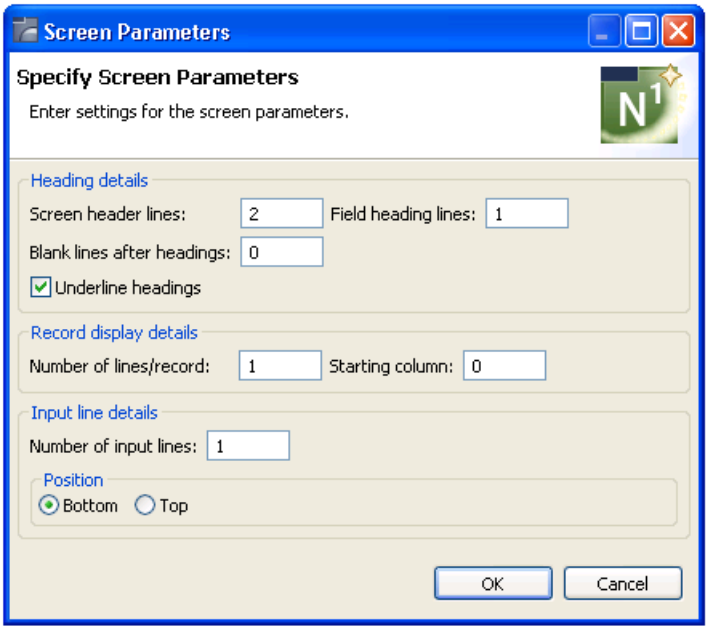
Specify Screen Parameters

This section describes the **Specify Screen Parameters** window, which defines how information is displayed on the generated screen.

To specify screen parameters

1 Select **Screen Parameters**.

The **Specify Screen Parameters** window is displayed. For example:



2 Define the following parameters:

Parameter	Description
Screen header lines	Number of screen heading lines displayed on the generated screen (by default, two panels).
Field heading lines	Number of field heading lines displayed on the generated screen (by default, one line is reserved for each field heading line).
Blank lines after headings	Number of blank lines inserted after the field heading lines. For example, if you specify "1", one blank line is inserted below each field heading line.
Underline headings	Indicates whether field headings are underlined. By default, this option is marked and field headings are underlined on the generated screen.

Parameter	Description
Number of lines/record	Number of screen lines required to display each record and its attributes (by default, one line is reserved for each record).
Starting column	Number of the screen column in which the selection column begins.
Number of input lines	Number of screen lines required to display the input keys (by default, one line is reserved for each input key).
Position Bottom	Indicates whether the input key lines are displayed at the bottom of the generated screen (by default, this option is selected and the input key lines are displayed at the bottom).
Position Top	Indicates whether the input key lines are displayed at the top of the generated screen. To have the input key lines displayed at the top of the screen, select this option.

3 Select **OK** to save the settings.

Specify Standard Parameters

The **Specify Standard Parameters** panel is displayed when a wizard is invoked; it is similar for all wizards. This section describes parameters on this panel that are common to multiple wizards. For information about parameters that are specific to individual wizards, see the applicable wizard section.

To specify standard parameters

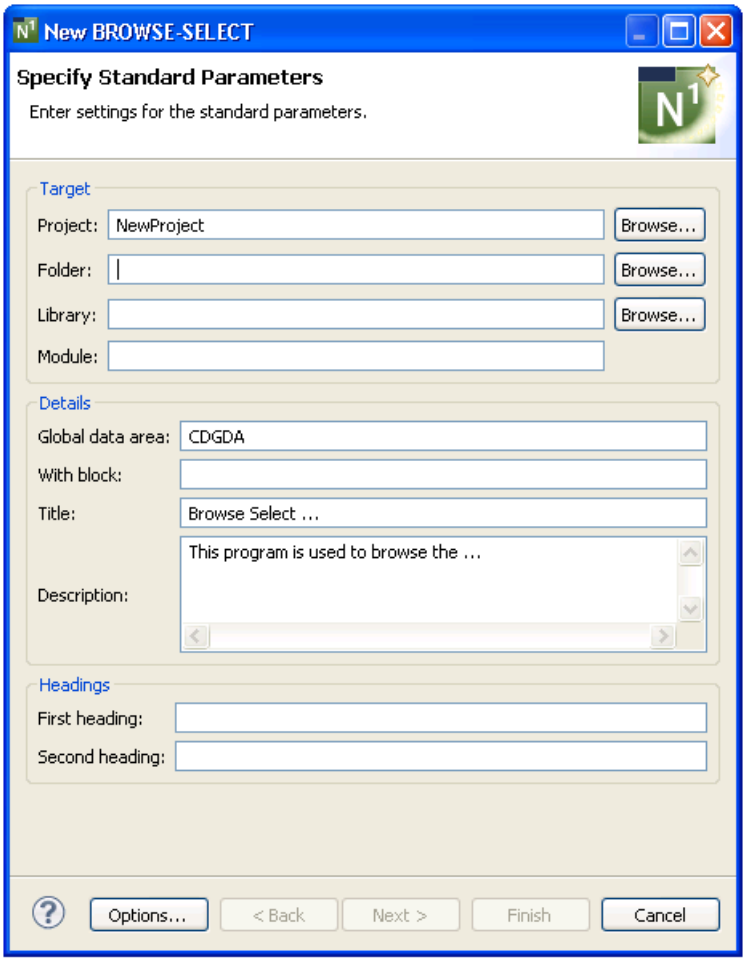
1 Open the context menu in the **Project Explorer** for the NaturalONE project in which you want to generate the modules.

Or:

Open the context menu in the **Project Explorer** for the library in which you want to generate the modules.

2 Select **Code Generation > New Using Construct Model >** *`WizardName`*.

The **Specify Standard Parameters** panel for the selected wizard is displayed. The following example shows the panel for the Browse-Select wizard:



3 Define the following parameters:

Parameter	Description
Library	Name of the library in which to store the generated modules. Either type the name or select **Browse** to display the available libraries for selection.
Module	Name of the module to be generated. This name must follow standard Natural naming conventions.
Global data area	Name of the global data area (GDA) used by the module to be generated. To allow inter-program communication, generated modules require a small number of global variables. The supplied CDGDA global data area contains the global variables required to test a generated module. Before creating a new application, copy this GDA from the SYSTEM library and rename it to match your naming conventions. Then add any additional global variables your application may require.
Title	Title for the module to be generated. You can customize the title for your application.
Description	Brief description of what the generated module will do. This field is used internally for documentation purposes.

Parameter	Description
First heading	First heading displayed on the generated panel. This heading is centered at the top of the generated panel and intensified.

Optionally, you can:

Task	Procedure
Define a different NaturalONE project in which to generate the module(s).	Type the name of the project in **Project** or select **Browse** to display a window listing the existing projects for selection. The project must currently exist.
Define a folder in which to generate the module(s).	Type the name of the folder in **Folder** or select **Browse** to display a window listing the existing folders for selection. The folder must currently exist within the selected NaturalONE project. **Note:** This option allows you to generate modules into more complex library structures (for example, "Natural-Libraries/*my library* (MYLIB)/SRC"). When this option is not specified, the modules will be generated into the basic library folder (for example, "Natural-Libraries/MYLIB/SRC", "Natural-Libraries/MYLIB/Subprograms", etc.).
Define a GDA block for use with the specified GDA.	Type the name of the GDA block in **With block**. You need only specify the lowest level block name; the corresponding path name is determined automatically. For more information about GDA blocks, see the Natural documentation.
Define a second heading for display on the generated panel(s).	Type the heading in **Second heading**. This heading is centered under the first heading and intensified.
Select generation options for the module(s).	Select **Options**. For information, see *Generation Options*.

4 Select **Next**.

The next panel for the specified wizard is displayed. For information, see the applicable wizard section.

Example of Generating a Program

This section provides an example of using the QUIT wizard to generate a quit program.

To generate a quit program

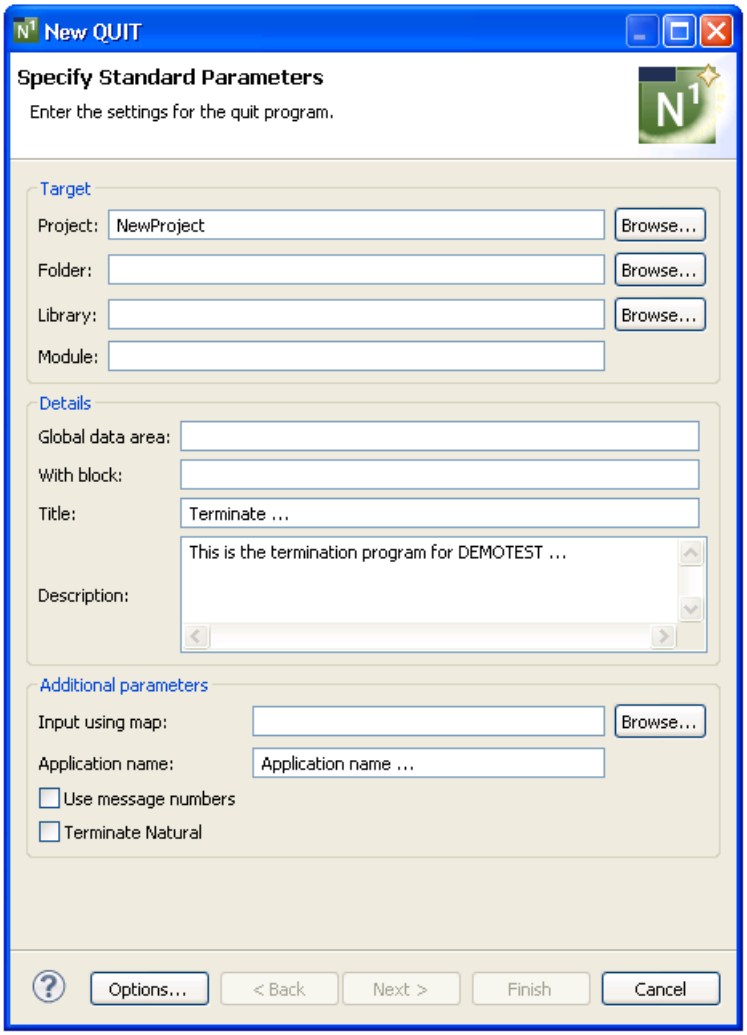
1 Open the context menu in the **Project Explorer** view for the NaturalONE project into which you want to generate the quit program.

Or:

Open the context menu in the **Project Explorer** view for the library into which you want to generate the quit program.

2 Select **Code Generation > New Using Construct Model > QUIT**.

First the **Progress Information** window is displayed, showing progress as the model specifications are initialized, and then the wizard panel is displayed. For example:



The names of the project and library from which this panel was invoked are displayed (you can change these if desired).

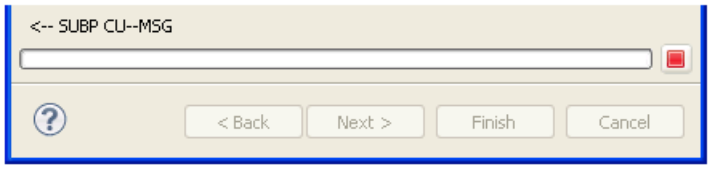
3 Type "MYQUIT" in **Module**.

4 Type "CDGDA" in **Global data area**.

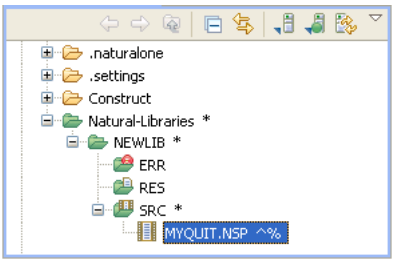
The CDGDA global data area contains the global variables required to test a generated module.

5 Select **Finish**.

The generation progress is displayed at the bottom of the panel. For example:



When generation is complete, the quit program is displayed in the **Project Explorer**. For example:



The quit program is also displayed in the editor. For example:

```
MYQUIT.NSP
* >Natural Source Header 000000
**SAG GENERATOR: QUIT                          VERSION: 5.3.1.9
**SAG TITLE: Terminate ...
**SAG SYSTEM: NEWLIB
**SAG GDA: CDGDA
**SAG DESCS(1): This is the termination program for DEMOTEST ...
**SAG APPLICATION-NAME: Application name ...
************************************************************************
* Program  : MYQUIT
* System   : NEWLIB
* Title    : Terminate ...
* Generated: Feb 28,2011 at 16:25:38 by PWRUSR
* Function : This is the termination program for DEMOTEST ...
*
*
*
* History
**SAG EXIT POINT CHANGE-HISTORY
************************************************************************
DEFINE DATA
GLOBAL USING CDGDA
LOCAL USING CDENVIRA /* Used to capture/restore previous environment.
LOCAL USING CDFLIPA  /* Used to change the KD lines.
LOCAL USING CDKEYLDA /* Used to set function keys + their alias.
LOCAL USING CDQTTXL  /* Text used by the Quit model.
LOCAL USING CDDIALDA /* Common local data.
LOCAL
01 #APPLICATION-NAME(A25) INIT <'Application name ...'>
01 #COMMENT-TXT1(A78)
01 #COMMENT-TXT2(A78)
01 #LAST-OBJECT(A8)   /* Object which initiated quit.
01 #LAST-PF-KEY(A4)
01 #RESUME-PF-KEY(A4)
01 #QUIT-INDEX(P3)    /* Index of quit key in CDKEYLDA table.
01 #USE-STANDARD-QUIT-KEY(L)
**SAG EXIT POINT LOCAL-DATA
END-DEFINE
*
FORMAT KD=ON
```

At this point, you can define additional processing within user exits. User exit code is preserved during regeneration. For information, see *Defining User Exits*.

Regenerate Natural Construct-Generated Modules

You can regenerate any module that was generated using a Natural Construct or client generation wizard. You can also select more than one project, folder, or object to regenerate multiple modules. When regenerating multiple modules, a selection window is first displayed to select the resources to be regenerated.

To regenerate Natural Construct-generated modules

1 Open the context menu for one or more projects, folders, or modules in the **Project Explorer**.

You can use standard selection techniques.

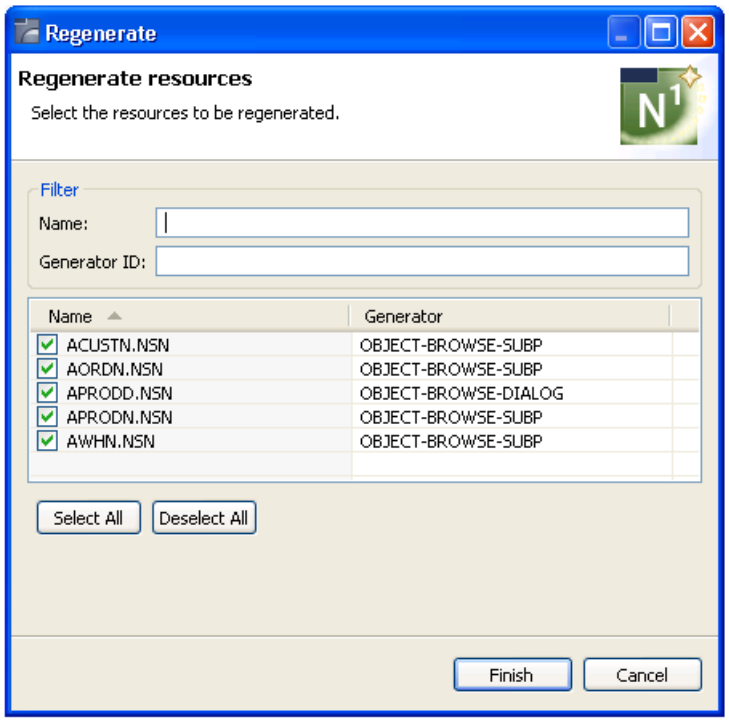
2 Select **Code Generation > Regenerate Using Wizard**.

The model PDA is uploaded to the mainframe to be populated and then downloaded to the local Eclipse environment to populate the PDA values before displaying the first wizard specification panel. You can then edit the specifications and select **Finish**.

Or:

Select **Code Generation > Regenerate**.

A progress window is displayed as the wizard locates and loads the regeneratable resources. Next, a selection window is displayed to choose the resources you want to regenerate. For example:



Using this panel, you can:

Task	Procedure
Filter the list of resources for selection.	Type a prefix in **Filter**. For example, if you type "AP", only the resources beginning with AP are selected.
Use a different code generator to regenerate the resource.	Type the generator ID in **Generator ID**.
Deselect all resources.	Select **Deselect All**.

After selecting the resources, select **Finish**. The modules are generated without displaying the wizard panels.

When regenerating a resource that was generated using a Natural Construct client generation wizard, the following process occurs:

1. The original generated source code is uploaded to the Natural server.

   This allows the `**SAG` lines to be parsed into the model PDA (and user exits to be processed later).
2. The Read program is executed to populate the model PDA.
3. The generation process begins, using the downloaded model PDA data.
4. User exits are merged on the server.
5. All generated modules are downloaded from the server to the client.

12 Natural Construct Administration

- Create a New Client Generation Wizard 208
- Download Natural Construct Resources to a Local Project 245
- Modify an Existing Natural Construct Resource 247
- Create and Maintain a Natural Construct Model 247
- Create and Maintain a Code Frame 251
- Create and Maintain a Natural Construct Model UI 258

This section describes how to use NaturalONE functionality to create and maintain Natural Construct resources.

Note: If you have customized any of the supplied modules on the server, you can either add them to the current project or add them to the downloaded Construct runtime project (which will overwrite the supplied modules). For more information, see *Add Customized Modules to the Construct Runtime Project*.

Create a New Client Generation Wizard

The most significant difference between Natural Construct in NaturalONE and on the server is the user interface. Instead of entering "NCSTG" and invoking the Modify server subprogram panels, a local wizard is used. This wizard is created using the model record file (.cstmdl extension), the model PDA, customizable XML files for the model UI file (.cstmdlui extension) and any reusable dialog UI files (.cstmdldg extension) or page UI files (.cstmdlpg extension), and a wizard engine. The engine calls the clear subprogram (called before the wizard is invoked and used to set defaults) or read subprogram (used to read the specifications for regeneration) on the server, displays the user interface and populates the PDA with data from the interface, and then calls the validation subprogram on the server to validate the user input. The model record data is identical to the data on the Natural server and must exist in the local environment. The wizard uses this data to determine the model PDA and clear, validate, and read subprograms for the model.

To create a new client generation wizard

1 Download the model record from the server installation.

   Note: For information on downloading the model record and PDA from the server, see *Download Natural Construct Resources to a Local Project*.

2 Use the model record to determine the name of the model PDA.

3 Download the model PDA from the SYSCST library.

4 Create the model UI file and, optionally, the reusable page UI and dialog UI files.

   These files include the main model UI file to map the user interface to the model PDA (.cstmdlui extension), as well as any reusable dialog UI files (.cstmdldg extension) or page UI files (.cstmdlpg extension).

The topics below describe how to create a client generation wizard for a model that has not been implemented locally.

The following topics are covered:

- User Interface (UI) File Examples

- Page Node
- Dialog Node
- Item Node
- GUI Controls
- Add a Tool Tip
- Set Up a Server Call
- Bind Data to GUI Controls
- Error Handling Tips for Field Names
- Generate NATdoc Documentation

User Interface (UI) File Examples

This section describes the user interface (UI) files used in the client generation framework.

The following topics are covered:

- Model UI File
- Reusable Dialog and Page UI Files

Model UI File

This section describes the model UI file (.cstmdlui extension). In the same way the model panels on the server relate to the model PDA, this file connects the client user interface with the model parameter data area (PDA) and/or specifications. On the client, these relationships are created using document nodes that associate the model PDA via the `pda:` keyword. or `specs:` keyword when the required input is not found in the model PDA. This information is then used during the generation process.

The following example illustrates the model UI file (.cstmdlui extension) for the supplied STARTUP wizard:

```
<model name="STARTUP" constructID="STARTUP" category="Construct">
     <version major="5" minor="3" release="1" />
     <description>Startup Model</description>
         <pages>
              <page id="StartupPage" title="Specify Standard Parameters">
                 <description>Enter settings for the startup program.</description>
                   <layout class="gridLayout" columns="3" />
              <children>
                   <group text="Target  ">
                        <layoutData class="gridLayoutData" horizontalSpan="3"
                             grabExcessHorizontalSpace="true" />
                        <layout class="gridLayout" columns="3" />
                        <children>
                        <!-- PROJECT -->
                             <label text="Project:" />
                                 <text id="ProjectTextText" text="{specs:project}"
                                       default="selection:project">
```

```
                                        <layoutData class="gridLayoutData"
                                             horizontalSpan="1" ↩
grabExcessHorizontalSpace="true" />
                                   </text>
                                   <cstBrowseProject />
                        <!-- LIBRARY -->
                             <label text="Library:" />
                                   <text id="LibraryText" text="{specs:library}"
                                        default="selection:library">
                                        <layoutData class="gridLayoutData"
                                             grabExcessHorizontalSpace="true" />
                                   </text>
                                   <cstBrowseLibrary />
                         <!-- MODULE -->
                              <label text="Module:" />
                                    <text id="ModuleText" text="{specs:module}">
                                        <layoutData class="gridLayoutData"
                                             grabExcessHorizontalSpace="true" />
                                    </text>
                              <label></label>
                        </children>
                    </group>
                    <group text="Details  ">
                         <layoutData class="gridLayoutData" horizontalSpan="3"
                              grabExcessHorizontalSpace="true" />
                         <layout class="gridLayout" columns="3" />
                         <children>
                         <!-- GDA -->
                             <label text="Global data area:" />
                              <text id="ModuleText" text="{pda:CUSTPDA.#PDAX-GDA}"
                                        default="CDGDA">
                                        <layoutData class="gridLayoutData"
                                             grabExcessHorizontalSpace="true" />
                                   </text>
                                   <cstBrowseNaturalObject text="Browse..."
                                        resultType="BrowseResults.NAME" ↩
fileExtension="NSG"
                                        result="{pda:CUSTPDA.#PDAX-GDA}" />
                         <!-- GDA with block -->
                              <label text="With block:" />
                                    <text id="GdaWithBlockText" ↩
text="{pda:CUSTPDA.#PDAX-GDA-BLOCK}">
                                         <layoutData class="gridLayoutData"
                                              grabExcessHorizontalSpace="true" />
                                    </text>
                              <label text="" />
                         <!-- TITLE -->
                              <label text="Title:" />
                                    <text id="TitleText" text="{specs:title}" ↩
default="Startup Program.">
                                         <layoutData class="gridLayoutData"
                                              grabExcessHorizontalSpace="true" />
```

```
                                    </text>
                               <label text="" />
                          <!-- DESCRIPTION -->
                               <label text="Description:" />
                                    <cstMulti id="DescriptionText" ↩
text="{pda:CUSTPDA.#PDAX-DESCS}"
                                        default="This is the main startup program ↩
....">
                                        <layoutData class="gridLayoutData"
                                         horizontalSpan="1" verticalAlignment="4" ↩
heightHint="60" />
                                    </cstMulti>
                               <label></label>
                          </children>
                     </group>
                     <!-- Group for SPECIFIC PARAMETERS -->
                     <group text="Specific parameters  ">
                          <layoutData class="gridLayoutData" horizontalSpan="3"
                               grabExcessHorizontalSpace="true" />
                          <layout class="gridLayout" columns="3" />
                          <children>
                          <!-- MAIN MENU PROGRAM -->
                               <label text="Main menu program:" />
                                    <text id="ModuleText" ↩
text="{pda:CUSTPDA.#PDAX-MAIN-MENU-PROGRAM}">
                                        <layoutData class="gridLayoutData"
                                             horizontalSpan="1" ↩
grabExcessHorizontalSpace="true" />
                                    </text>
                                    <cstBrowseNaturalObject text="Browse..."
                                        resultType="BrowseResults.NAME" ↩
fileExtension="NSP"
                                    result="{pda:CUSTPDA.#PDAX-MAIN-MENU-PROGRAM}" ↩
/>
                          <!-- QUIT PROGRAM -->
                               <label text="Quit program:" />
                                    <text id="ModuleText" ↩
text="{pda:CUSTPDA.#PDAX-QUIT-PROGRAM}">
                                        <layoutData class="gridLayoutData"
                                             horizontalSpan="1" ↩
grabExcessHorizontalSpace="true" />
                                    </text>
                                    <cstBrowseNaturalObject text="Browse..."
                                          resultType="BrowseResults.NAME" ↩
fileExtension="NSP"
                                       result="{pda:CUSTPDA.#PDAX-QUIT-PROGRAM}" />
                          <!-- COMMAND PROCESS -->
                               <label text="Command processor:" />
                                    <text id="CommandText" ↩
text="{pda:CUSTPDA.#PDAX-PROCESSOR}">
                                        <layoutData class="gridLayoutData"
                                             horizontalSpan="1" ↩
```

```
grabExcessHorizontalSpace="true" />
                                 </text>
                              <button style="SWT.CHECK" text="Error transaction ↩
processing"
                                     selection="{pda:CUSTPDA.#PDAX-ERROR-TA}">
                                     <layoutData class="gridLayoutData"
                                          horizontalSpan="3" ↩
grabExcessHorizontalSpace="true" />
                                 </button>
                           </children>
                     </group>
                 </children>
           </page>
      </pages>
</model>
```

Reusable Dialog and Page UI Files

The Natural Construct client generation framework supports reusable dialogs and pages. The dialog UI or page UI code can be created once as a separate file and then included in multiple model UI files. In this way, changes to the page or dialog UI code will be automatically reflected in any client generation wizard that includes that page or dialog UI file.

Model UI File

The following example illustrates a model UI file (.cstmdlui extension) for MYMODEL, which includes reusable page and dialog UI files:

```
<model name="MYMODEL" constructID="MODELA" category="Construct">
      <version major="5" minor="3" release="1" />
      <description> This model demonstrates reusable page/dialogs.
      </description>
      <pages>
            <page id="Page1" title="Page1 title">
                  <description>Enter settings for this non-reusable ↩
page.</description>
                  <children>
                        ... UI nodes go here ...

                        <cstDialogButton id="button" text="Show a dialog...">
                              <dialog include="MyReusableDialog">
                                    <replacements>
                                          <stringReplacement ↩
target="%DIALOG_PDA_NAME%"
                                                replacement="NEWPDA" />
                                    </replacements>
                              </dialog>
                        </cstDialogButton>

                        ... more UI nodes go here ...
                  </children>
```

```
            </page>
            <page include="MyReusablePage">
                  <replacements>
                        <stringReplacement target="%PAGE_PDA_NAME%"
                              replacement="NEWPDA" />
                  </replacements>
            </page>
      </pages>
</model>
```

Dialog UI File

The following example illustrates the reusable dialog UI file (.cstmdldg extension) for MYMODEL:

```
<modelUIDialog>
      <dialog windowTitle="My Dialog" title="My dialog title" message="My dialog ↩
message.">
            <children>
                  <button style="SWT.CHECK" text="My button"
                        selection="{pda:%DIALOG_PDA_NAME%.#PDA-BOOLEAN-FIELD}">
                  </button>

                  ... more UI nodes go here …

            </children>
      </dialog>
</modelUIDialog>
```

Page UI File

The following example illustrates a reusable page UI file (.cstmdlpg extension) for MYMODEL:

```
<modelUIPage>
      <page id="Page2" title="Page2 title">
             <description>Enter settings for this reusable page.</description>
             <children>
                   <text id="MyText" text="{pda:%PAGE_PDA_NAME%.#PDA-FIELD}" ↩
required="true" />

                   ... UI nodes go here ...

                   <cstDialogButton id="button" text="Show a dialog...">
                         <dialog include="MyReusableDialog">
                               <replacements>
                                     <stringReplacement target="%DIALOG_PDA_NAME%"
                                           replacement="NEWPDA" />
                               </replacements>
                         </dialog>
                   </cstDialogButton>

                   ... more UI nodes go here …
```

```
            </children>
      </page>
</modelUIPage>
```

Page Node

The **page** node represents a page displayed through the wizard. Within this node, child nodes represent SWT GUI controls. This section describes the elements and attributes you can define within the **page** node.

- Description
- HelpID
- ID
- Include
- Optional
- Replacements
- Title

Description

When defined, this attribute provides a brief description of the page. For example:

```
page id="StartupPage" title="Specify Standard Parameters">
     <description>Enter settings for the startup program.</description>
```

HelpID

When defined, this attribute enables the help button on the page and links the Eclipse help system to the applicable help ID. For example:

```
<page id="StParms" title="Specify Standard Parameters" helpID="StParmsHelpID">
```

ID

This attribute defines the name of the page. For example:

```
page id="StartupPage" title="Specify Standard Parameters">
```

Include

When defined, this attribute defines the name of a reusable page used by the model. For example:

```
<page include="MyReusablePage">
```

Optional

When defined as true, this attribute disables validation for the page; if the remaining pages are also defined as optional, the **Finish** button will be enabled. For example:

```
<page id="2"  title="TWO" optional ="true">
```

By default, a page is not optional (i.e., it is mandatory).

Tip: To allow users to select **Finish** when only optional pages remain, group all optional pages together at the end of the XML file.

Replacements

When defined, this element defines nested stringReplacement elements that allow simple string replacements to be performed (for example, PDA bindings) for a reusable page. For example:

```
<replacements>
      <stringReplacement target="pda:MYPDA" replacement="pda:MYPDA2" />
</replacements>
```

In this example, all occurrences of "pda:MYPDA" in the reusable page will be replaced with "pda:MYPDA2" when a client generation wizard imports and uses the page.

Title

This attribute defines the title displayed on the page.

Dialog Node

The **dialog** node represents a dialog that will be displayed to a user. For example:

```
<dialog windowTitle="Edit Row" title="Row Details" message="Enter row details">
     <children>
         <label text="&amp;Code:" />
              <text id="Oms11" text="{pda:CUMNPDA.#PDA-CODE}">
                    <layoutData class="gridLayoutData" horizontalSpan="2"
                         grabExcessHorizontalSpace="false">
                    </layoutData>
              </text>
         <label text="&amp;Function:" />
              <text id="Oms12" text="{pda:CUMNPDA.#PDAX-FUNCTIONS}">
                   <layoutData class="gridLayoutData" horizontalSpan="2"
```

```
                        grabExcessHorizontalSpace="true">
                    </layoutData>
                </text>
            <label text="&amp;Program name:" />
                <text id="Omsb13" text="{pda:CUMNPDA.#PDA-PROGRAM-NAME}">
                    <layoutData class="gridLayoutData" horizontalSpan="1"
                        grabExcessHorizontalSpace="false" />
                    <style type="naturalObject" />
                </text>
                <cstBrowseNaturalObject text="Browse..."
                    fileExtension="NSP" result="{pda:CUMNPDA.#PDA-PROGRAM-NAME}">
                </cstBrowseNaturalObject>
        </children>
</dialog>
```

This node is similar to the **page** node; any XML control that can be defined for a page can also be defined for a dialog; in addition, any stringReplacement can be defined for a reusable dialog. The following attributes are defined within the **dialog** node:

Attribute	Description
`include`	Name of a reusable dialog (only used when the dialog parent node is cstDialogButton). For information, see *Reusable Dialog and Page UI Files*.
`message`	Banner text displayed on the dialog.
`title`	Title displayed on the dialog.
`windowTitle`	Internal name used for the dialog.

Note: If you do not select **OK** to close the edit window, the edits will not be applied (i.e., the values in the table will not change).

Item Node

The **item** node represents a value/display combination to map between the text displayed on a GUI control and the actual underlying value. For example:

```
<items>
      <item value="1" display="Winter"/>
      <item value="2" display="Spring"/>
      <item value="3" display="Summer"/>
      <item value="4" display="Fall"/>
</items>
```

The following attributes are defined within the **item** node:

Attribute	Description
`value`	Value that will be saved internally.
`display`	Value that will be displayed on the page.

GUI Controls

GUI controls are represented by nodes in the client generation framework and are used to bind user input from the interface to associated fields in the model PDA or specifications. Before the first specification panel is displayed, the model's clear subprogram on the server is invoked and typically populates default values in the PDA. These default values will be presented to the user (unless they have been overridden at the XML node level).

All controls have the following traits in common:

- Any property of an SWT (Standard Widget Toolkit) control can be modified with an attribute in the XML file. The XML attribute in the XML must map to a Get/Set method in the corresponding SWT control.
- All controls have an ID attribute. Whenever there is an error with a control, the ID attribute will be displayed to assist in diagnosing the problem. Although there is no checking for duplicate ID attributes, it is highly recommended that each control have a unique ID.
- All controls support the tool tip option, which provides information about its use.

Note: For more information, see ***Bind Data to GUI Controls*** and ***Default Properties Applied to GUI Controls***.

This section covers the following topics:

- Button
- Combo
- Composite
- cstCombo
- cstDeriveServerButton
- cstDialogButton
- cstRadioGroup
- cstTable
- dateTime
- Group
- Label
- Text Box
- Multi-Line Text Box

- Browse Button Controls

Button

This GUI control is a button the user can select to initiate an action.

Attributes

Attribute	Description
`style`	An SWT constant indicating the style of the button. For example, SWT.CHECK will create a check box.
`text`	Text displayed on the button.
`selection`	Boolean binding value indicating whether the button is selected or not. If the button is a check box, this attribute indicates whether the button is marked or not.
`onWidgetSelected`	Link to an event to be handled.

Example

```
<button style="SWT.CHECK" text="Check" selection="{pda:CUSTPDA.#PDAX-ERROR-TA}"/>
```

In this example, the value of the checked box (either true or false) will be placed in the #PDAX-ERROR-TA Boolean field in the CUSTPDA parameter data area for the STARTUP model.

Combo

This GUI control is a drop-down list that allows the user to either enter text or select a value from a list of available choices.

Attributes

Attribute	Description
`text`	Binding property used to bind the text property for the combo box to an underlying field.
`values`	A list of items to display in the combo box, separated by commas.

Example

```
<combo id="myCombo" text="{pda:level1.selection}" values="A,B,C,D" />
```

In this example, the user will be able to select A, B, C, or D as an input value for the specified PDA field. The field cannot be blank.

Note: If this is not a required field, add a 5th entry to represent a blank (for example, values=" ,A,B,C,D").

Composite

This GUI control is an invisible control that hosts other controls.

Child Nodes

Node	Description
layout	Defines settings for the layout strategy to use.
children	Defines the child controls.

Example

```
<composite>
  <layout class="gridLayout" columns="2" />
  <children>
    <label text="Array 1" />
    <text text="{pda:level1.Array(1)}" />
    <label text="Array 2" />
    <text text="{pda:level1.Array(2)}" />
  </children>
</composite>
```

When the number of columns are defined for a table, the cells are filled from left to right and top to bottom. In this example, there will be 2 columns and 2 rows where the first row is ARRAY 1 and the data is in array(1).

cstCombo

This section describes the **cstCombo** control. This control is similar to the **combo** control, except it allows one value to be displayed on a panel and a second, different value to be stored in the PDA.

Attributes

Attribute	Description
`value`	Bound field that stores the value of the combo box.
`defaultValue`	Value to use when the bound field is not set.

Child Nodes

Node	Description
item	For information, see *Item Node*.

Set Default Values

Unless otherwise specified, the default value is the first item in the list. When a blank is not acceptable, you can provide a default value. For example:

```
<cstCombo value="{pda:#INPUT.VALUE}"  defaultValue="1">
      <items>
            <item value="1" display="Ontario"/>
            <item value="2" display="Quebec"/>
      </items>
</cstCombo>
```

When a blank is acceptable, you can provide an item for blanks. For example:

```
<cstCombo value="{pda:#INPUT.VALUE}">
      <items>
            <item value="" display=""/>
            <item value="1" display="Ontario"/>
            <item value="2" display="Quebec"/>
      </items>
</cstCombo>
```

Note: The `value` attribute is used, as opposed to the `text` attribute, to ensure that the number (for example, "1") goes into the PDA, while the text in `display` (for example, "Ontario") is displayed on the wizard panel.

cstDeriveServerButton

This section describes the **cstDeriveServerButton** control, which calls a subprogram to derive data from the server.

Note: For information on using the cstDeriveServerButton control to refresh defaults in the model PDA or derive values from the server, see *Set Up a Server Call*. For example, you can use this control if I wanted to start my first browse row on line 3 and there are 2 lines of input how many rows could I fit on the screen…the number 3 and 2 the user enters in and the button goes off and calculates the number of rows that fit on the screen) values for the model PDA.

Attributes

Attribute	Description
`style`	An SWT constant indicating the style of the button. For example, SWT.PUSH will create a button.
`text`	Text displayed on the button.
`serverCallID`	Call ID for the server call that defines how the data from the proxy PDA (the fields on the server) gets populated into the model PDA (the fields on the client) and vice versa. For more information, see *Set Up a Server Call*.

Example

```
<cstDeriveServerButton style="SWT.PUSH"
     text="Refresh Default Methods" serverCallID="Default_Methods">
</cstDeriveServerButton>
```

cstDialogButton

This section describes the **cstDialogButton** control, which displays a custom dialog when selected.

Attributes

Attribute	Description
`text`	Text displayed on the button.

Child Nodes

Node	Description
dialog	For information, see *Dialog Node*.

Example

```
<cstDialogButton id="button" text="Display Options...">
     <dialog windowTitle="Display Options" title="Display Options"
          message="Select options for display">
          <children>
               <label text="&amp;Option Code:" />
                    <text id="Oms11" text="{PDA:#INPUT.VALUE}">
                    </text>
          </children>
     </dialog>
</cstDialogButton>
```

cstRadioGroup

This section describes the **cstRadioGroup** control, which creates a group box containing radio buttons (one radio button for each **item** child node).

Attributes

Attribute	Description
`value`	Bound field that stores the value of the group box.
`defaultValue`	Value to use when the bound field is not set.
`orientation`	Alignment of the radio buttons. Valid values are horizontal or vertical (the default).
`text`	Text displayed on the group box.

Child Nodes

Node	Description
item	For information, see *Item Node*.

Example

```
<cstRadioGroup value="{pda:#INPUT.VALUE}" text="Province" defaultValue="2" ↩
orientation="horizontal">
     <items>
          <item value="1" display="Ontario"/>
          <item value="2" display="Quebec"/>
     </items>
</cstRadioGroup>
```

cstTable

This section describes the **cstTable** control, which provides a table (grid) for an array.

Notes:

1. The bindings in a dialog do not include the array index. The selected row will be added to the table when the **Edit** button is selected.
2. You can create a table that uses multi dimensions of an array. For an example of this functionality, see the Object-Browse-Subp wizard.

Attributes

Attribute	Description
countField	Name of the field that stores the actual count of rows (©# field). If the count field is not specified, the table will use the number of elements declared in the array as the row count. For example, A (A10/1:10) will have 10 rows displayed in the table.
editOnly	When this attribute is set to true, the rows in cstTable can only be edited (i.e., the Add and Delete buttons will be disabled). By default, rows in cstTable can be added, deleted, and edited.
maxRows	Maximum number of rows displayed for the table. By default, this number is the upper bound of the array field.
numberColumnDisplay	Column header for the row number column; the default is "#".
numberColumnWidth	Column width; the default is 25 pixels.

Cascading Deletes

If the first dimension of an array is defined as a Group field and the second dimension is defined as a group under the first dimension, cstTable will automatically cascade deletes to the second dimension. In the following example, if Parent-Row(5) is deleted, Child-Row(5,*) will be deleted as well:

```
01 Parent-Row (1:10)
 02 Field2 (A10)
 02 Field3 (A10)
 02 Child-Row (1:5)
  03 Child1 (A10)
  03 Child2 (A10)
```

Nested Array Indexes

You can nest array indexes within a table. For example, you can create a table on one dialog that contains the first dimension of an array and pass the indexes for a second and third dimension to a second table in another dialog. For example:

```
<statement display="DREW" id="DREW" velocityTemplate="ESCAPE.vm">
      <pages>
            <page id="Start" title="ESCAPE statement">
                  <description>Enter statement options</description>
                  <layout class="gridLayout" columns="1" />
                  <children>
                        <cstTable>
                              <layoutData class="gridLayoutData" ↩
horizontalSpan="1"grabExcessHorizontalSpace="true" />
                              <columns>
                                 <column fieldName="A" display="A" width="75" ↩
/>
                                 <column fieldName="B" display="B" width="50" ↩
/>
                              </columns>
```

```
                              <dialog>
                                    <layout class="gridLayout" columns="2" />
                                    <children>
                                          <label text="A:" />
                                          <text text="{pda:A}">
                                                <layoutData ↩
class="gridLayoutData" horizontalSpan="1"grabExcessHorizontalSpace="true" />
                                          </text>
                                          <label text="B:" />
                                          <text text="{pda:B}">
                                                <layoutData ↩
class="gridLayoutData" horizontalSpan="1"grabExcessHorizontalSpace="true" />
                                          </text>
                                          <label text="C(n,1):" />
                                          <text text="{pda:C(1)}">
                                                <layoutData ↩
class="gridLayoutData" horizontalSpan="1"grabExcessHorizontalSpace="true" />
                                          </text>
                                          <label text="C(n,2):" />
                                          <text text="{pda:C(2)}">
                                                <layoutData ↩
class="gridLayoutData" horizontalSpan="1"grabExcessHorizontalSpace="true" />
                                          </text>
                                    </children>
                              </dialog>
                        </cstTable>
                  </children>
            </page>
      </pages>
      <pda>
      <![CDATA[
         01 Group1(1:3)
          02 A (A) DYNAMIC
          02 B (A) DYNAMIC
          02 Group2 (1:2)
           03 C (A) DYNAMIC
           03 D (A) DYNAMIC]]>
      </pda>
</statement>
```

In this example, the user selected the second row of the table for editing and the first index (2) is populated; the C field will be generated as C(2,1) and C(2,2).

Group a Nested Table

You can group a nested table, which allows you to place a border around the table and provide a title for the group. For example:

```
<group text="Window location">
      <layoutData class="gridLayoutData"horizontalSpan="2" ↩
grabExcessHorizontalSpace="true" />
      <children>
            <cstTable id="table">
```

```
                  <layoutData class="gridLayoutData"horizontalSpan="3" ↩
grabExcessHorizontalSpace="true"grabExcessVerticalSpace="true" ↩
horizontalAlignment="SWT.FILL"verticalAlignment="SWT.FILL" />
                  <columns>
                        <column ↩
fieldName="CUOMPDA.#PDAX-SCROLL-LINE"display="Line" width="150" />
                        <column ↩
fieldName="CUOMPDA.#PDAX-SCROLL-COL"display="Column" width="150" />
                  </columns>
            </cstTable>
      </children>
</group>
```

Child Nodes

columns

This node defines attributes for a column within the cstTable control. These attributes are:

Attribute	Description
`fieldName`	Name of the field in the PDA to which the column is bound.
`display`	Heading used for the column.
`width`	Width of the column. The default is 25 pixels.

`blankTest`	Used to detect empty rows within a table. A row is considered empty when all the flagged columns for the row are empty. By default, all columns within a row are flagged as being part of the blank test. To un-flag a column, set the `blankTest` attribute to "false".
	
	A field within a specific row/column is considered empty using the following rules for the value:
	■ Numeric fields (INPF): zero (0, 0.0, 00.00, etc.)
	■ Any other field is blank when its string representation is blank or only contains whitespace ("", " ", "<tab>", etc.)
	
	In the following example, only the first column is used to determine whether a row is empty:
	
	`<cstTable>`
	`<columns>`
	`<column fieldName="A" display="A" width="75" />`
	`<column fieldName="B" display="B" width="50" blankTest="false"/>`
	`<column fieldName="C" display="C" width="50" blankTest="false" />`
	`</columns>`
	`</cstTable>`

dialog

For information, see *Dialog Node*.

Example

The following example illustrates a cstTable control for the supplied Menu wizard:

```
<cstTable id="table" countField="CUMNPDA.#PDA-TOTAL-MENU-LINES">
  <layoutData class="gridLayoutData" horizontalSpan="3"
      grabExcessHorizontalSpace="true" grabExcessVerticalSpace="true"
      horizontalAlignment="SWT.FILL" verticalAlignment="SWT.FILL" />
  <columns>
    <column fieldName="CUMNPDA.#PDA-CODE" display="Code" width="100" />
    <column fieldName="CUMNPDA.#PDAX-FUNCTIONS" display="Function"
        width="100" />
    <column fieldName="CUMNPDA.#PDA-PROGRAM-NAME" display="Program Name"
        width="100" />
  </columns>
  <dialog windowTitle="Edit Row" title="Row Details" message="Enter row details">

    <children>
      <label text="&amp;Code:" />
      <text id="Oms11" text="{pda:CUMNPDA.#PDA-CODE}">
        <layoutData class="gridLayoutData" horizontalSpan="2"
            grabExcessHorizontalSpace="false"
        >
        </layoutData>
      </text>
      <label text="&amp;Function:" />
      <text id="Oms12" text="{pda:CUMNPDA.#PDAX-FUNCTIONS}">
        <layoutData class="gridLayoutData" horizontalSpan="2"
            grabExcessHorizontalSpace="true"
        >
        </layoutData>
      </text>
      <label text="&amp;Program name:" />
      <text id="Omsb13" text="{pda:CUMNPDA.#PDA-PROGRAM-NAME}">
        <layoutData class="gridLayoutData" horizontalSpan="1"
            grabExcessHorizontalSpace="false" />
         <style type="naturalObject" />
      </text>
        <cstBrowseNaturalObject text="Browse..."
            fileExtension="NSP" result="{pda:CUMNPDA.#PDA-PROGRAM-NAME}"
        >
        </cstBrowseNaturalObject>
    </children>
  </dialog>
</cstTable>
```

dateTime

This GUI control is an edit control that accepts a date and/or time, and optionally presents a drop-down calendar (depending on the SWT style). For information, see:

Class DateTime.

Attributes

Attribute	Description
`value`	Binding value indicating the location in which to store and retrieve the date value.

Example

```
<dateTime style="SWT.DATE \| SWT.DROP_DOWN \| SWT.BORDER " ↩
value="{pda:level1.date}"/>
```

In this example, the user is restricted to only entering a date; this value will go into the specified field in the model PDA.

Group

This GUI control is a rectangular border/frame that groups related controls and has a group heading on its border.

Attributes

Attribute	Description
`text`	Text displayed on the group box border.

Child Nodes

Node	Description
layout	Settings for the layout strategy to use.
children	Node under which the child controls are placed.

Example

```
<group text="Arrays">
  <layout class="gridLayout" columns="2" />
  <children>
    <label text="Array 1" />
    <text text="{pda:level1.Array(1)}" />
    <label text="Array 2" />
    <text text="{pda:level1.Array(2)}" />
  </children>
</group>
```

Note: For more examples of group boxes, such as the target group, see the startup.xml example in *Model UI File*.

Label

This GUI control is an SWT (Standard Widget Toolkit) label used to display text.

Note: A label control does not receive focus nor generate input events.

Attributes

Attribute	Description
`text`	Text displayed for the label.

Example

```
<label text="Library:" />
```

In this example, the label is used as the prompt for the Library input field. For more examples of labels, see the startup.xml example in *Model UI File*.

Text Box

This GUI control is a text box that allows users to input a single line of text. The text control can also be read-only.

Note: As text fields have no description, define a label control to describe their purpose.

Attributes

Attribute	Description
`text`	Binding property indicating how to bind the text in a text box to an underlying data source. For example: `<text id="LibraryText" text="{specs:library}" required="false">`
`default`	Default value shown when the text box is first displayed.
`required`	Boolean value indicating whether the text box must be defined. When this attribute is set to true, an error message will be displayed and the user will not be allowed to navigate off the page until the text is filled in.
`displayName`	When the required attribute is set to true and the field is blank, this name will be displayed as the field name in the error message. For example: `"<displayName> cannot be blank"`

Attribute	Description
	Note: If displayName has not been specified, the control ID will be used as the field name in the error message.

SWTstyle	Indicates the Eclipse (SWT) style for the text box, such as scroll bars or multi-line. The following example binds a non-array field to a multi-line text box with scroll bars:
	`<text text="{pda:LOGICAL-CONDITION}" id="condition"`
	`SWTstyle="SWT.MULTI | SWT.BORDER | SWT.V_SCROLL | ↩`
	`SWT.H_SCROLL">`
	`<layoutData class="gridLayoutData"`
	`grabExcessHorizontalSpace="true" heightHint="40" />`
	`</text>`

Child Nodes

style

The style node allows you to specify the style of the text box. The valid styles are:

Style	Description

maxLength	Maximum number of characters the text box will accept. For example:
	`<text id="ModuleText" text="{specs:Module}">`
	`<style maxLength="20"/>`
	`</text>`

case	Converts the user input into a particular case. The valid values are: "upper", "lower", or "mixed" (no conversion takes place). For example:
	`<text id="ModuleText" text="{specs:Module}">`
	`<style case="upper"/>`
	`</text>`

numbersOnly	Boolean value indicating whether only numbers 0-9 will be accepted. For example:
	`<text id="ModuleText" text="{specs:Module}">`
	`<style numbersOnly="true" />`
	`</text>`
	Note: Signs (+/-) cannot be used.

numericOnly	Boolean value indicating whether only numeric keys will be accepted. For example:
	`<text id="ModuleText" text="{specs:Module}">`
	`<style numericOnly="true" />`
	`</text>`
	Note: Signs (+/-) can be used.

Style	Description

type	Value indicating a combination of styles. Possible values are "naturalObject" and "naturalFieldName". For example:
	`<text id="ModuleText" text="{specs:Module}">`
	`  <style type="naturalObject"/>`
	`</text>`

Multi-Line Text Box

This GUI control is a multi-line text box that can be bound to an array of string fields.

Attributes

Attribute	Description
`text`	Binding property indicating how to bind the text in a text box to an underlying data source. For example: `<cstMulti id="LibraryText" text="{specs:library}" required="false">`
`default`	Default value shown when the text box is first displayed.
`required`	Boolean value indicating whether the text box must be defined. When this attribute is set to true, an error message will be displayed and the user will not be allowed to navigate off the page until the text is filled in.
`displayName`	When the `required` attribute is set to true and the field is blank, this name will be displayed as the field name in the error message. For example: `"<displayName> cannot be blank"` **Note:** If displayName has not been specified, the control ID will be used as the field name in the error message.

Example

```
<cstMulti id="DescriptionText" text="{pda:CUSTPDA.#PDAX-DESCS}" required="true">
```

In this example, #PDAX-DESCS is an array in CUSTPDA.

Browse Button Controls

This section describes the standard **Browse** button controls, which are used in combination with the edit field (text box) controls whenever an existing object is referenced within a wizard. The edit field is used to enter the name of an existing object; the **Browse** button is used to browse and select the object from a list of all possible choices.

The standard **Browse** button controls are:

Message Number

This button displays a dialog to select a message number. The selected message text is stored in the location indicated by the `result` attribute.

By default, cstBrowseMessage creates a **Browse** button that, when selected, will search the Natural SYSERR library on the server and return all error messages associated with the Natural Construct application library (CSTAPPL). The user can search for messages in other languages or libraries by changing the input values, or can start browsing at a different error number. The user can then select a message to populate the location specified in the `result` attribute.

Attributes

Attribute	Description
`result`	Binding attribute indicating where to store the selected message text. If no view or **Cancel** is selected, the bound field will not change from its previous value.

Example

```
<cstBrowseMessage result="{pda:level1.Module}"/>
```

Natural Library

This button displays a dialog to select a Natural library in the local environment. The button is bound to the ModelSpecs library property.

Attributes

Attribute	Description
`allowDefault`	Boolean value indicating whether the ModelSpecs library value should be set based on the current user selection in the Eclipse environment. For example, if the user selects a Natural library (or descendant of the library), that library name can be used as the default value.

Example

```
<cstBrowseLibrary allowDefault="true"/>
```

Note: Since all generated code must be stored in a Natural library, you can define this node to use the current user library as the default library.

Natural Object

This button displays a dialog to select one or more Natural objects in the local environment. The list of modules can be restricted to a specified module type. The selected value is used to populate the location specified by the `result` attribute.

Attributes

Attribute	Description
`fileExtension`	File extension used to limit the available selections.
`result`	Binding attribute indicating where to store the user selection.

Module Types and Extensions

Module Type	Extension
DDM	NSD
GDA	NSG
Helproutine	NSH
LDA	NSL
Map	NSM
PDA	NSA
Program	NSP
Subprogram	NSN
Text	NST

Example

```
<cstBrowseNaturalObject text="Select PDA" fileExtension="NSA" ↩
result="{pda:level1.PDA}" />
```

This example illustrates a **Browse** button for PDA files.

Natural Project

This button displays a dialog to select a Natural project. The selected project is stored in the location indicated by the ModelSpecs project property.

Attributes

Attribute	Description
`allowDefault`	Boolean value indicating whether the ModelSpecs project value should be set based on the user selection in the Eclipse environment. For example, if the user selects a Natural project (or descendant of the project), that name can be used as the default value.

Example

```
<cstBrowseProject allowDefault="true"/>
```

In this example, the name of the current Natural project is used to populate the project name in the model specifications.

Predict Field

This button displays a dialog to browse all the fields in a DDM and select a field from the previously selected Predict view. The selected field is stored in the location indicated by the `result` attribute.

Attributes

Attribute	Description
`autoClear`	Clears the bound result when the associated view value changes. This will prevent a view from not containing the desired field.
`descriptorsOnly`	Attribute indicating whether the Descriptors only field on the field selection panel is selected by default and only descriptor fields will be displayed. When true, the Descriptors only field is selected by default; when false or not specified, the Descriptors only field is not selected.
`result`	Binding attribute indicating where to store the selected Predict field. If no field or **Cancel** is selected, the bound field will not change from its previous value.
`view`	Binding attribute indicating the view for which to list fields.

Example

```
<cstBrowsePredictField autoClear="true"
  view="{pda:CUBOPDA.#PDAX-PRIME-FILE}" ↩
result="{pda:CUBOPDA.#PDAX-PHYSICAL-KEY(1,1)}"
  descriptorsOnly="true"/>
```

Predict View

This button displays a dialog to select a Predict view. The selected view is stored in the location indicated by the `result` attribute.

Attributes

Attribute	Description
result	Binding attribute indicating where to store the selected Predict view. If no view or **Cancel** is selected, the bound field will not change from its previous value.

Example

```
<cstBrowsePredictView result="{pda:level1.#PDAX-PRIME-FILE}"/>
```

Add a Tool Tip

A tool tip provides information about using a control when the cursor is moved over the control. All SWT controls have a tool tip text property and all XML control nodes support the tool tip option. For example:

```
<text id="MyTextID" toolTipText="Tool tip ↩
text" displayName="displayName" required="true" text="{pda:Mypda.MyField}"/>
```

Set Up a Server Call

While the wizard's clear subprogram provides default values for the model PDA when the wizard is started, values that the user specifies, such as the file name, can be used to derive more information. The derived information, however, requires a server call, which can be made when a wizard panel is left (onLeave) or entered (onEnter) or via a button. For example, after the user selects **Next** on a wizard panel, a subprogram can be called to fill in the appropriate values on the subsequent panel. This is particularly useful when input data on the first panel (for example, the name of the object-browse subprogram or file) is required to derive data for the second screen.

This section describes the two methods used to set values on wizard panels.

- Set Values Whenever a Panel is Entered or Left
- Set Values Whenever a Button is Selected

Set Values Whenever a Panel is Entered or Left

A subprogram can be called whenever a wizard panel is left (onLeave) or entered (onEnter) to provide values for the model PDA. For obvious reasons, the onEnter event will never be called on the first page. Similarly, the onLeave event will never be called on the last page. In all cases, the server call must be defined.

This section covers the following topics:

- Definitions
- Server Calls
- Field Mappings

- onLeave and onEnter Events

Definitions

Term	Description
Model PDA	Parameter data area associated with the model; it contains fields used for the user interface (i.e., the PDA specified in the .cstmdl file for the model).
Proxy	Subprogram on the server that is used to serialize data for any subprogram that was not generated by the CST-PROXY model.
Proxy PDA	Parameter data area associated with the subprogram called by the proxy subprogram; it contains fields used to input data into the model PDA fields.

Server Calls

The onEnter and onLeave events and the cstDeriveServerButton control call serverCalls, which are defined as child nodes within the model node in the XML.

Attribute	Description
`id`	Unique identifier of the server call. This ID is used to identify which server call to invoke from an onLeave or onEnter event or button control, which in turn identifies which subprogram proxy to execute on the server.
`pdas`	A comma-delimited list of client text files that represents the definitions found in the PDA for the subprogram the proxy calls.
`proxyName`	The name of the proxy subprogram to invoke on the server.

The following example illustrates a serverCall to provide default methods for the Object-Browse-Select-Subp wizard:

```
<model name="OBJECT-BROWSE-SELECT-SUBP" constructID="OBJECT-BROWSE-SELECT-SUBP"
category="Construct"
     >
     <serverCalls>
          <serverCall id="Default_Methods" pdas="WTCBUDEF-inlinePDA,CSASTD"
           proxyName="WTPBUDEF">
     >
```

where:

- `WTPBUDEF` is a proxy subprogram (generated with the CST-PROXY model) used to access the WTCBUDEF subprogram.

  WTCBUDEF has the following parameters:

  ```
  DEFINE DATA
    PARAMETER
    01 #INPUTS
  ```

```
    02 #OBJECT-BROWSE (A8)
  01 #OUTPUTS
    02 #METHOD-MAPPING (1:15)
      03 #BROWSE-KEY (A32)
      03 #BROWSE-COUNT (L)
      03 #METHOD-NAME (A32)
    02 #METHOD-MAPPING-COUNT (N2)
  PARAMETER USING CSASTD
  LOCAL USING CUBUPDA
END-DEFINE
```

- `CSASTD` is a .NSA file on the client that contains the parameter data area (PDA) definitions for the CSASTD PDA (standard messaging parameters used by all models). It passes messages between subprograms and is typically used for error handling.
- `WTCBUDEF-inlinePDA` is a .NSA file on the client that contains all the other variables:

```
01 #INPUTS
  02 #OBJECT-BROWSE (A8)
01 #OUTPUTS
  02 #METHOD-MAPPING (1:15)
    03 #BROWSE-KEY (A32)
    03 #BROWSE-COUNT (L)
    03 #METHOD-NAME (A32)
  02 #METHOD-MAPPING-COUNT (N2)
```

In the serverCall example, the `pdas` attribute defines both .NSA files above. For example:

```
pdas="WTCBUDEF-inlinePDA,CSASTD"
```

Field Mappings

Each server call can have one or more field mappings. Field mappings define how data is copied from and to the model and proxy PDAs. The attributes for the field mappings are:

Attribute	Description
`modelField`	Name of the field in the model PDA to be copied.
`proxyField`	Name of the field in the proxy PDA to be copied.
`direction`	Determines when the field will be copied. Valid values are: ■ in Field will be copied from the model PDA to the proxy PDA before the call to the server is made (for example, the name of the object-browse subprogram used by an Object-Browse-Select-Subp wizard). ■ out

Attribute	Description
	Field will be copied from the proxy PDA to the model PDA immediately after the server call is made (for example, the methods derived from the object-browse subprogram). ■ in_out Field will be copied from the model PDA to the proxy PDA before the server call and from the proxy PDA to the model PDA after the server call.

The following example illustrates the field mappings for the serverCalls example above:

```
<serverCall id="Default_Methods" pdas="DefaultMethods,CSASTD"
proxyName="WTPBUDEF">
     <mappings>
          <mapField modelField="CUBUPDA.#PDAX-OBJECT-BROWSE"
               proxyField="#INPUTS.#OBJECT-BROWSE" direction="in" />
          <mapField modelField="CUBUPDA.#PDAX-BROWSE-KEY" ↩
proxyField="#OUTPUTS.#BROWSE-KEY"
               direction="out" />
          <mapField modelField="CUBUPDA.#PDAX-BROWSE-COUNT"
               proxyField="#OUTPUTS.#BROWSE-COUNT" direction="out" />
          <mapField modelField="CUBUPDA.#PDAX-METHOD-NAME"
               proxyField="#OUTPUTS.#METHOD-NAME" direction="out" />
     </mappings>
```

The following example illustrates a sample of code from the .cstmdlui file for the Object-Browse-Select-Subp wizard:

```
<model name="OBJECT-BROWSE-SELECT-SUBP" constructID="OBJECT-BROWSE-SELECT-SUBP"
     category="Construct">
          <serverCalls>
               <serverCall id="Default_Methods" pdas="DefaultMethods,CSASTD"
                    proxyName="WTPBUDEF">
                   <mappings>
                        <mapField modelField="CUBUPDA.#PDAX-OBJECT-BROWSE"
                             proxyField="#INPUTS.#OBJECT-BROWSE" direction="in" />
                        <mapField modelField="CUBUPDA.#PDAX-BROWSE-KEY" ↩
proyxField="#OUTPUTS.#BROWSE-KEY"
                             direction="out" />
                        <mapField modelField="CUBUPDA.#PDAX-BROWSE-COUNT"
                            proxyField="#OUTPUTS.#BROWSE-COUNT" direction="out" />
                        <mapField modelField="CUBUPDA.#PDAX-METHOD-NAME"
                          proxyField="#OUTPUTS.#METHOD-NAME" direction="in_out" />
                        <mapField modelField="CUBUPDA.#PDA-METHOD-MAPPING-COUNT"
                             proxyField="#OUTPUTS.#METHOD-MAPPING-COUNT" ↩
direction="out" />
                   </mappings>
               </serverCall>
     </serverCalls
     <onLeave serverCallID="Default_Methods" schedule="FIELD_CHANGED"
         fieldNames="CUBUPDA.#PDAX-OBJECT-BROWSE" />
```

onLeave and onEnter Events

Once the server call has been defined, it can be connected to:

- The **Next** button via the onLeave event
- The **Back** button via the onEnter event
- A user-defined button via the cstDeriveServerButton control

To eliminate unnecessary calls to the server, the onLeave and onEnter events contain a `schedule` attribute that can be set to only call the server when required.

Note: This option is not available for the cstDeriveServerButton control, as it is assumed that a server call will always be required when this button is selected.

When navigating from one page to another (i.e., by selecting **Next**), the order of events are:

1. Current page onLeave event.
   - Copy the contents of the model PDA to the proxy PDA using the input mappings.
   - Issue a CALLNAT statement to the server.
   - Copy the contents of the proxy PDA to the model PDA using the output mappings.
2. Next page onEnter event.
   - Copy the contents of the model PDA to the proxy PDA using the input mappings.
   - Issue a CALLNAT statement to the server.
   - Copy the contents of the proxy PDA to the model PDA using the output mappings.
3. Show next page.

Note: Selecting **Back** on the wizard page has no effect; onLeave and onEnter are only invoked when **Next** is selected.

The onLeave and onEnter events are defined as child nodes within the page node in the XML. These events specify which subprogram will be called whenever the page is entered or left. The attributes for these events are:

Attribute	Description
`serverCallID`	Call ID for the server call that defines how the data from the proxy PDA (the fields on the server) gets populated into the model PDA (the fields on the client) and vice versa.
`schedule`	Determines when the CALLNAT will be issued. Values are "ALWAYS", "FIELD_CHANGED", "FIRST_TIME_ONLY". **Note:** This functionality does not apply to a button.

Attribute	Description
fieldNames	When schedule is set to "FIELD_CHANGED", this attribute provides a comma-delimited list of fields in the PDA that the user may have changed. If the user does change one of the fields, the subprogram will be called.

The following example illustrates the onLeave event:

```
<page id="StdParms" title="Specify Standard Parameters"
helpID="com.softwareag.naturalone.gen.doc.code.2acgw100"
     >
     <description>Enter settings for the standard parameters.
     </description>
     <onLeave serverCallID="Default_Methods" schedule="FIELD_CHANGED"
     fieldNames="CUBUPDA.#PDAX-OBJECT-BROWSE" />
```

When the user selects **Next** on the wizard panel, the subprogram (identified by the serverCallID attribute) retrieves the method names, key names, and count.

Set Values Whenever a Button is Selected

Server data can be derived using the cstDeriveServerButton control in the client generation framework. When the user selects this button on a wizard panel, the appropriate subprogram is called to derive data from the server. Use this GUI control whenever user input is required.

The following example creates a button called **Refresh Default Methods**:

```
<cstDeriveServerButton style="SWT.PUSH"
     text="Refresh Default Methods" serverCallID="Default_Methods">
</cstDeriveServerButton>
```

Whenever the user selects **Refresh Default Methods** on the wizard panel, the subprogram (identified by the serverCallID attribute) is called to retrieve the method names, key names, and count.

Note: For more information, see *cstDeriveServerButton*.

Bind Data to GUI Controls

Within the XML file for a client generation wizard, certain nodes represent the GUI controls to be created for the screen. To allow data from the parameter data area (pda) or specification (specs) object to be bound to a GUI control, you can specify what data and which default values to display for the control.

Notes:

1. To ensure consistency within the defaulting methods used on the client and the server, set the default values for the PDA in the model's clear subprogram on the server. The clear subprogram

is invoked before the first panel of the wizard is displayed, so typically there is no need to set default values in the XML file for the client generation wizard.

2. The standard Eclipse SWT controls are used; you can set any property for a control that has a corresponding Set/Get method. To determine which properties are available, refer to the Eclipse documentation.

The following example illustrates how to bind the `Mypda.MyField` PDA field to a text box control:

```
<text id="MyTextID" text="{pda:Mypda.MyField}"/>
```

In this example, `text="{pda:Mypda.MyField}"` indicates that the text property for the GUI control is bound to the PDA field called Mypda.MyField, where Mypda is the level 1 structure within the model PDA and MyField is typically a #PDAX field name within the model PDA. Any changes to the GUI will be automatically reflected in the underlying PDA field.

Note: For this example, the GUI control must have a text property. If not, an error is displayed.

The notation to bind data to a GUI control is:

```
<source>:<binding>[=<default>]
```

where:

- `source` is the pda or specs keyword
- `binding` is the name of a field or property with which to bind

  Notes:

  1. When binding to a specs object, the field or property must have a corresponding Set/Get method.
  2. The field name is typically fully qualified (i.e., *`level1.fieldName`*.
- `default` is the default value to display for a field.

  The default value (typically set in the model's clear subprogram on the server) has two possible notations: one for the current selection and one for data settings on the dialog.

  Note: If the default value is set in the model's clear subprogram, the =<default> notation is not required in the XML; if the =<default> notation is specified, this value will override the value set in the clear subprogram.

This section covers the following topics:

- Use Logical Data to Enable or Disable Controls
- Override Default Values

- Separate Default Attributes for GUI Controls
- Default Properties Applied to GUI Controls
- Default Selection Notation
- Default Dialog Settings Notation
- Examples of Binding Notations

Use Logical Data to Enable or Disable Controls

You can bind a control property to a logical field in a parameter data area (PDA).

Example

Assume the following PDA settings:

```
01 FIELDS
02 LOGICAL (L)
02 TEXT (A) DYNAMIC
```

And the following syntax:

```
<text text="{pda:level1.TEXT}" enabled = "!{pda:level1.LOGICAL}"/>
```

In this example, when the LOGICAL field is true, the enabled property for the text control will be false (i.e., the text field will be disabled).

Override Default Values

Although default values for the PDA are typically set by the model's clear subprogram on the server, they can be overridden on the client by a value from cache or by directly assigning a value. If more than one method is used, the value taken is the last one assigned in the following order:

1. From the model's clear subprogram on the server.
2. From the dialog: notation.
3. From the direct assignment of the default value.

Separate Default Attributes for GUI Controls

Some GUI controls have a separate `default` attribute, which can be expressed as:

```
<text text="{pda:MY.FIELD=A}"/>
```

or

```
<text text="{pda:MY_FIELD} default="A"/>
```

The GUI controls with separate `default` attributes are: combo box, button, text box, and multiline text box.

Default Properties Applied to GUI Controls

The following table describes which properties the `default` attribute applies to each GUI control:

GUI Control	Default Property
Button	Selected
Check box	Selected
Combo box	Text
Text box	Text

Default Selection Notation

When a wizard is started from the **Project Explorer**, an item is usually selected in the view. Depending on which item is selected, you can set default values for GUI controls. For example, the name of a Natural library on the wizard panels can be defaulted to the name of the library selected in the view.

Note: If the default value cannot be determined, the value will not be set for the control.

The notation to define the default value for a view selection is:

```
<source>:<binding>=selection:<selection Type>
```

where `selection Type` is the default value to display for the field. The selection types are:

Selection Type	Default Value
natProject	Name of the selected Natural project.
project	Name of the selected project.
container	Name of the selected container (package).
file	Name of the selected file.
library	Name of the selected Natural library.
extension	Name of the selected file with the specified extension. To define this default value, add the following notation: `extension=NSN` where `NSN` is the extension used for the selected file. The selected file must contain the specified file extension.

For example, the following notation defines a checkbox that is bound and defaulted to unchecked:

```
<button style="SWT.CHECK" selected="{pda:level1.MyBoolean}" default="false"/>
```

Default Dialog Settings Notation

Data a user has previously specified for a GUI control can be saved and then reloaded as a default value for the control, which eliminates the need for users to enter repetitive information. Each piece of saved data is stored as a key value pair, where both the key and the value are strings. The notation is:

```
<source>:<binding>=dialog:<key>
```

where `key` contains the default value to display for the field.

For example, the following notation defines a text box that is bound and defaulted to the My.Dialog.Key dialog setting:

```
<text id="MyText" text="{pda:level1.MyField}" enabled="{pda:level1.MyBoolean}" ↩
default="{dialog:My.Dialog.Key}"/>
```

Examples of Binding Notations

The following table illustrates examples of binding notations:

Example	Will bind the GUI control to:
`pda:INPUT.NAME="FRED"`	A PDA field called INPUT.NAME with a default value of FRED.
`specs:project=selection:project`	The project property for the specs object and use the selected project name as the default value.
`specs:module=selection.extension="NSN"`	The module property for the specs object and use the selected file for the default value when its extension is .NSN.
`pda:INPUT.PREFIX=dialog:"My.Dialog.KEY"`	A PDA field called INPUT.PREFIX and use the value stored in My.Dialog.KEY in the dialog settings as the default value (i.e., the last value entered in this field when the wizard was last invoked).

Error Handling Tips for Field Names

When the validation subprogram for a client generation wizard returns a field in error, the fieldMatchHint attribute can be used to provide a "hint" to "match" the error field to one or more redefined fields. This allows for the scenario where a field is redefined and the wrong field is returned. You can also match multiple fields by separating each one with a comma. For example:

```
<text id="Predict view is required" required="true"
            text="{pda:CUSCPDA.#PDAX-PRIME-FILE}" ↩
fieldMatchHint="FIELDS.DUMMY_BEFORE_REAL_FIELD,CUSCPDA.#PDAX-PFILE">
```

Note: Although the fieldMatchHint applies to all bindable controls, do not use the fieldMatchHint attribute with buttons. Typically, buttons are associated with a text box and focus should be set on the text box instead of the button.

When errors are encountered, the following search order is used to find the bound field corresponding to the field in error:

- Field match with index
- Field match without index
- Field match with hint

Generate NATdoc Documentation

If the NATdoc option is enabled in the supplied CSXDEFLT subprogram, the code generated by the client generation wizards will include a user exit containing the name of the author, as well as the date and time the module was generated. Once this exit has been added to the module, it must be manually maintained. For example:

```
**SAG DEFINE EXIT NAT-DOCS
/** :author PWRUSR -- Generated Feb 14,2011 at 10:09:17
**SAG END-EXIT
```

In addition, NATdoc comments will be added to the external PDAs. For example:

```
DEFINE DATA
   PARAMETER USING ACUSTK  /** :in /* Search key values
   PARAMETER USING ACUSTD  /** :out /* Returned row data
   PARAMETER USING ACUSTP  /** :inout /* Restricted data
   PARAMETER USING CDBRPDA  /** :inout /* Generic browse object parms
   PARAMETER USING CDPDA-M  /** :out /* Msg info                        ↩
```

The runtime modules on the client also contain external PDAs containing NATdoc comments. For example:

```
DEFINE DATA PARAMETER
/* >Natural Source Header 000000
/* :Mode S
/* :CP
/* <Natural Source Header
 1 CDHASHA /** used in object maint to calculate hash value
   2 #FUNC (I4) /** :inout not required
   2 #CTX (B156) /** :inout not required
   2 #TEXT (A) DYNAMIC /** :in data to be hashed
END-DEFINE
```

NATdoc documentation can then use this information to generate documentation based on your selection. For example, you can generate NATdoc documentation for all the modules in a library.

Notes:

1. For information about CSXDEFLT, refer to *Code Generation* > *Using Natural Construct* > *Natural Construct Administration* > *Create a New Client Generation Wizard* > ***Enable NATdoc Generation***.
2. For information about NATdoc, see *Using NaturalONE* > *Generating API Documentation with NATdoc*.

Download Natural Construct Resources to a Local Project

To create a new client generation wizard or customize an existing model or code frame, you must download the resources from the server.

To download Construct resources to your local project

1 Locate the Natural Construct installation on the server.

Note: Natural Construct must be installed on the server.

2 Open the context menu for the **Construct** root node.

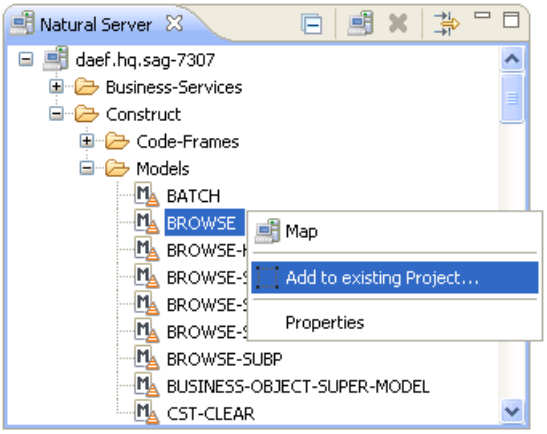
Or:

Expand the root node and select one or more model and/or code frame nodes or files using standard selection techniques.

Note: Children of the selected nodes are automatically included in the download (for example, selecting the **Models** root node will download all models from the server).

3 Select **Add to existing Project**.

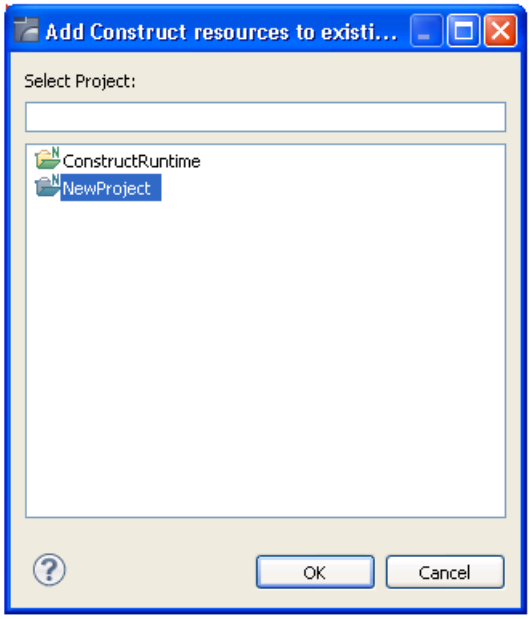
For example:



A list of available projects is displayed.

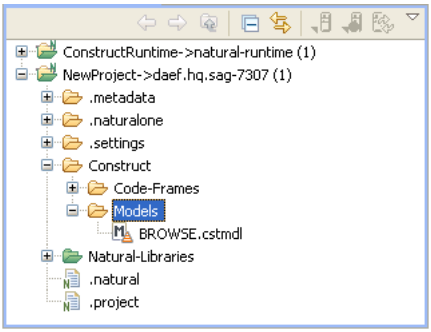
4 Select the project into which you want to download the models (or code frames).

For example:



5 Select **OK**.

A progress window is displayed as the model record is downloaded from the **Natural Server** view to the local project in the **Project Explorer**. Expand the **Construct** root node to display the downloaded resources. For example:



Modify an Existing Natural Construct Resource

This section describes how to modify an existing Natural Construct resource from your server installation, such as a model or code frame, in the Eclipse environment. To modify existing models and/or code frames:

- Download the resource from your Natural Construct installation on the server to your local environment. For information, see ***Download Natural Construct Resources to a Local Project***.
- Modify the resource as desired. For information, see ***Modify an Existing Model***, ***Modify an Existing Code Frame,*** or ***Modify an Existing Model UI.***
- Upload the modified resource to the server using standard NaturalONE functionality.

Create and Maintain a Natural Construct Model

This section describes how to create and maintain a Natural Construct model.

The following topics are covered:

- Create a New Model
- Modify an Existing Model Record

Tip: For deeper details on Natural Construct modeling, go to **https://documentation.softwareag.com/** and find the documentation of *Natural Business Services* > *Natural Construct Administration and Modeling*.

Create a New Model

To create a new Natural Construct model

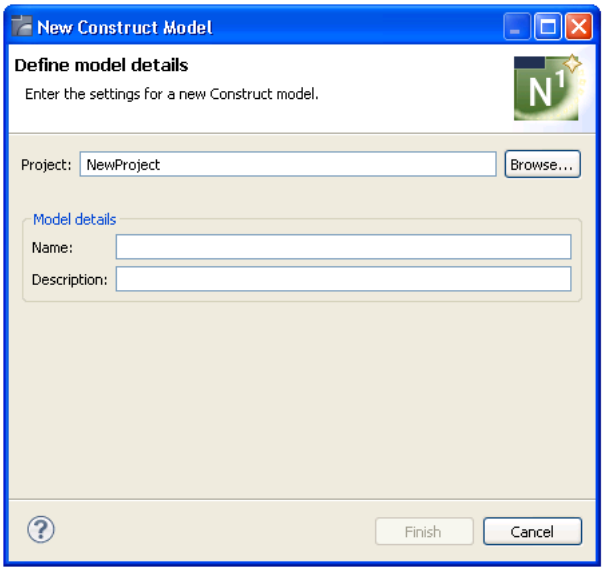
1 Open the context menu in the **Project Explorer** for the NaturalONE project into which you want to generate the model.

Or:

Open the context menu in the **Project Explorer** for the **Construct** root node or **Construct > Models** node into which you want to generate the model.

2 Select **Code Generation > New Construct Model**.

The **Define model details** panel is displayed. For example:



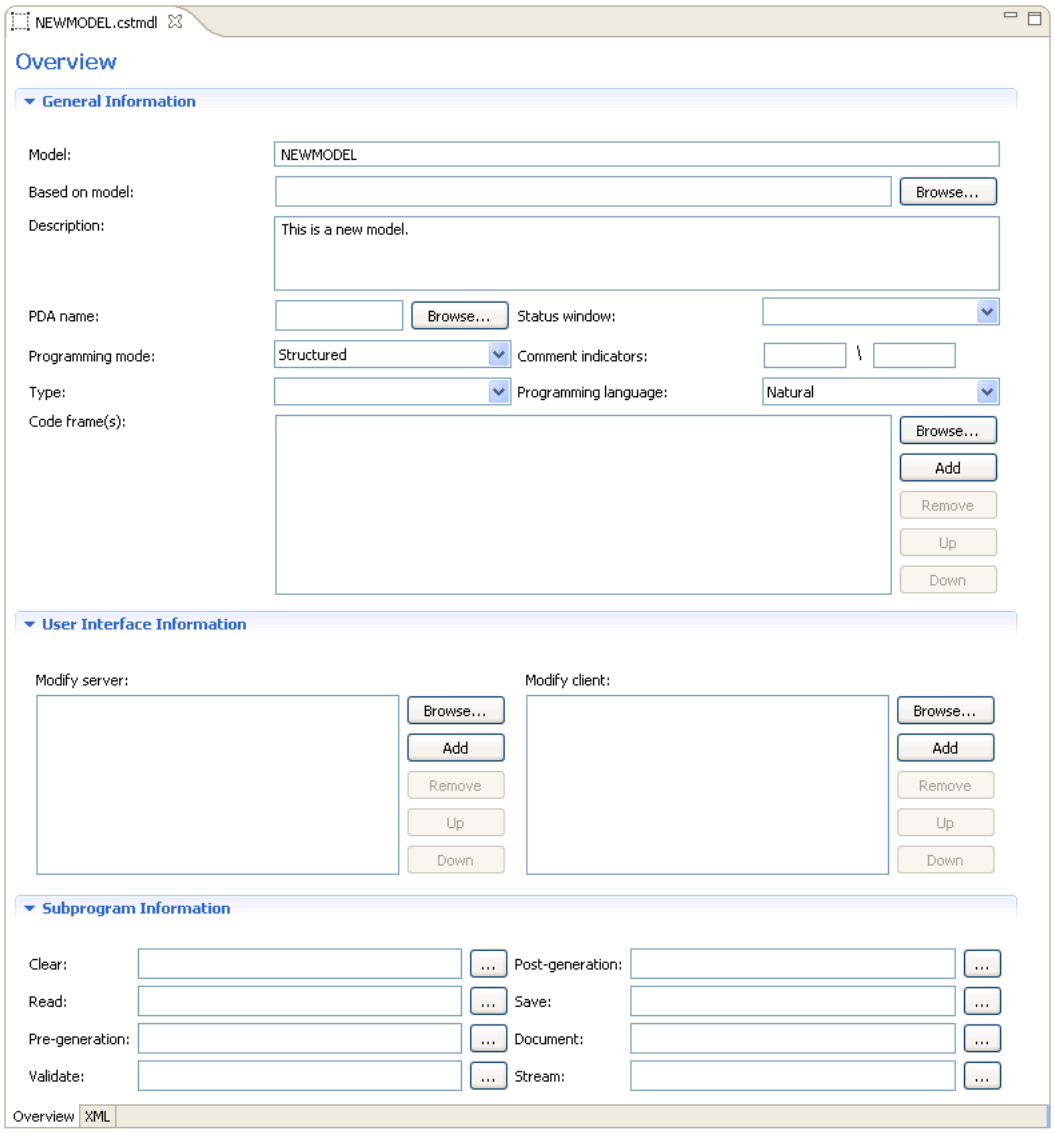
Note: To change the name of the project, type the name of a new project in **Project** or select **Browse** to display the available projects for selection.

3 Type the name of the new model UI file in **Name**.

4 Type a brief description of the model in **Description**.

5 Select **Finish**.

The new model file is listed in the **Project Explorer** and displayed in the editor. For example:



6 Specify the parameters for the new model.

For more information on these parameters go to **https://documentation.softwareag.com/** and find the documentation of *Natural Business Services* > *Natural Construct Administration and Modeling* > *Creating New Models*.

7 Create the model UI file (.cstmdlui extension), as well as any reusable dialog UI files (.cstmdldg extension) or page UI files (.cstmdlpg extension).

For information, see ***Create a New Client Generation Wizard***..

Note: You can only create the XML file after the PDA has been downloaded from the server. For information, see ***Download Natural Construct Resources to a Local Project***.

8 Upload the new model to your Natural Construct installation on the server.

Modify an Existing Model Record

A model is comprised of the following components:

- Model record file (.cstmdl extension)
- One or more XML files to define the user interface.

  For example, the model UI file (.cstmdlui extension) and any reusable dialog UI files (.cstmdldg extension) or page UI files (.cstmdlpg extension).
- Model PDA

  Tip: To modify the model PDA, first download the model record and determine the name of the PDA. Once you know the name, you can download the PDA from the SYSCST library on the server, modify it in the local editor, and then upload it to the SYSCST library.

- Code frames
- Subprograms

This section describes how to modify a Natural Construct model record in the local environment.

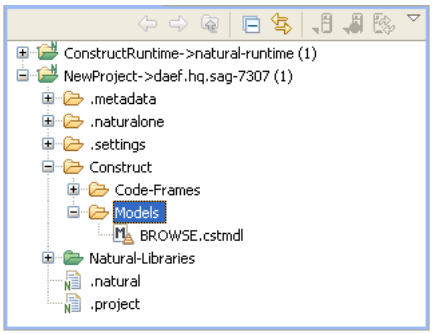
To modify an existing model record

1 Download the model record from the **Natural Server** view.

  For information, see *Download Natural Construct Resources to a Local Project*.

2 Expand the **Construct > Models** root node in the **Project Explorer**.

  For example:



3 Open the model file (.cstmdl extension) in the editor.

4 Modify the model record as desired.

For information about the editor, see *Create a New Model*.

5 Save the model record changes.

6 Upload the model record to the Natural Construct server installation using standard NaturalONE functionality.

Create and Maintain a Code Frame

This section describes how to create and maintain a code frame.

The following topics are covered:

- Create a New Code Frame
- Modify an Existing Code Frame
- View a Code Frame in the Outline View

Create a New Code Frame

This section describes how to create a new code frame.

The following topics are covered:

- Use the Code Frame Editor
- Create the Code Frame

Use the Code Frame Editor

To create a new code frame, use the NaturalONE code frame editor to replicate the code frame data defined in the standard editor on the server. The NaturalONE code frame editor uses a combination of special $ variables and line text to represent the server editor columns and input boxes (for example, type codes, condition codes, description input box, etc.).

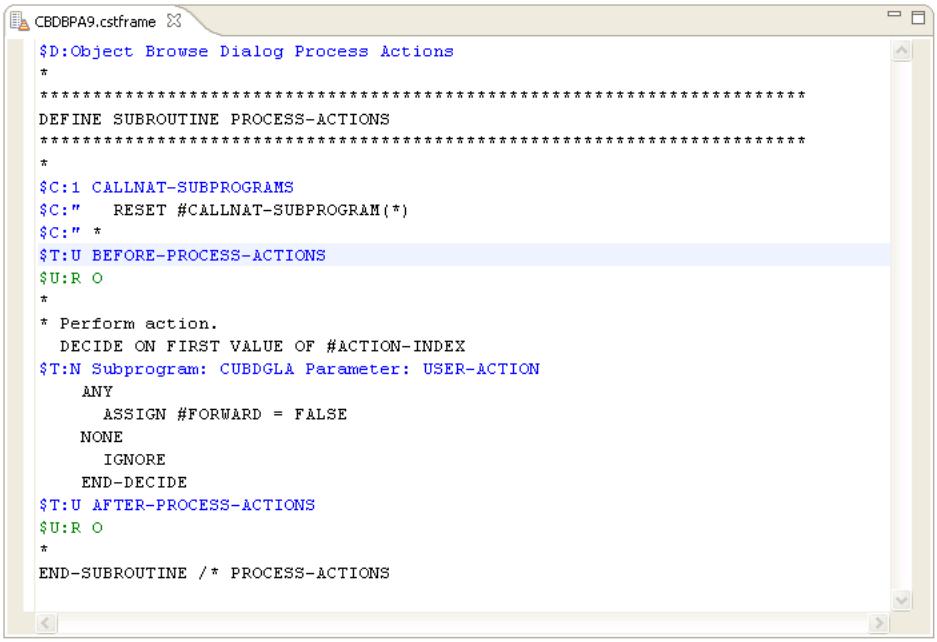
The editor allows three types of code frame lines:

Line	Description	Text Color
$D:*n*	Contains a description of the code frame, where *n* is a description of up to 45 characters (equivalent to the description line on the server). Maximum of one $D line per code frame.	Blue
$U:*n*	Contains details about user exits included within the code frame (equivalent to the user exit edit window on the server), where: ■ $U:R R (User exit required when a value is specified after :R) ■ $U:G *n* (Generate as subroutine when a value is specified after :G) ■ $U:S SAMP (Sample subprogram ..... SAMP)	Green

Line	Description	Text Color
	■ $U:U GUISAMP (GUI sample subprogram .. GUISAMP) ■ $U:L *n* (Default user exit code that requires one $U:L line for each default line)	
{$C:*n*} {$T:*n*} TEXT	Contains text and optional fields, where { } indicates optional fields and: ■ $C:*n*, where *n* indicates the condition code level in the editor on the server. Valid condition code levels are: ■ 1–9 Indicates a new condition for this level. The conditions are Boolean combinations of the condition constants specified for the generator. If the condition specified on the line is True, all subsequent code with quotation marks (") is included in the generated program. **Tip:** Every $C:*n* is equivalent to starting another IF statement. ■ $C:" Indicates that text on this line is a continuation of the previous block of code and subject to the last condition specified. ■ $C:blank Indicates that the corresponding line is constant text and is included unconditionally. ■ $T:*n*, where *n* indicates the line type in the editor on the server. Valid line types are: ■ N Indicates a subprogram (the text on this line must follow the format: `Subprogram:name {Parameter:name}`, where { } indicates a parameter is not required (similar to code frames on the server). ■ U Indicates a user exit ■ F Indicates a nested code frame ■ * Indicates a comment line within the code frame, which will not be generated into code ■ B Indicates a blank line ■ X	Black (inserted directly into the generated program, based on the $C: value) Blue (represents logic, such as user exit names, subprogram/parameter names, boolean values, etc., and will not be inserted directly into the generated program)

Line	Description	Text Color
	Indicates a conditional user exit. The line must also contain a corresponding $C:*n* entry to be valid. ■ TEXT represents the code frame logic and maps to columns 1-72 in the standard editor on the server (maximum 72 characters).	

The following example illustrates the CBDBPA9 code frame defined in the local code frame editor:



```
CBDBPA9.cstframe
$D:Object Browse Dialog Process Actions
*
************************************************************************
DEFINE SUBROUTINE PROCESS-ACTIONS
************************************************************************
*
$C:1 CALLNAT-SUBPROGRAMS
$C:"   RESET #CALLNAT-SUBPROGRAM(*)
$C:" *
$T:U BEFORE-PROCESS-ACTIONS
$U:R O
*
* Perform action.
  DECIDE ON FIRST VALUE OF #ACTION-INDEX
$T:N Subprogram: CUBDGLA Parameter: USER-ACTION
    ANY
      ASSIGN #FORWARD = FALSE
    NONE
      IGNORE
    END-DECIDE
$T:U AFTER-PROCESS-ACTIONS
$U:R O
*
END-SUBROUTINE /* PROCESS-ACTIONS
```

For more information on creating and editing code frames go to **https://documentation.software-ag.com/** and find the documentation of *Natural Business Services* > *Natural Construct Administration and Modeling* > *Using the Code Frame Editor*.

Create the Code Frame

To create a new code frame

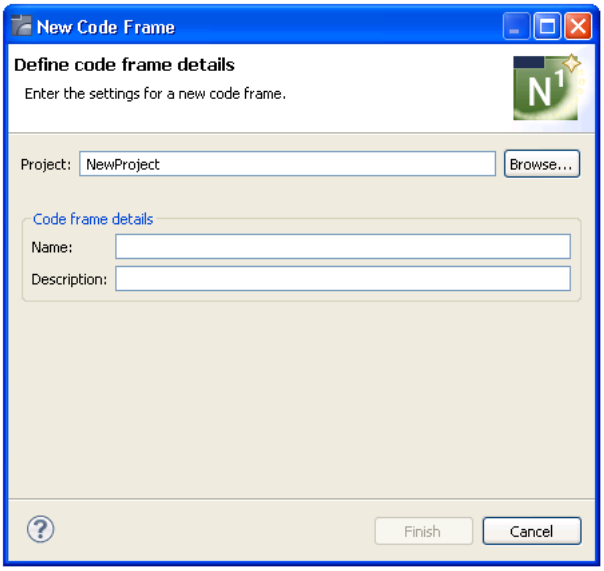
1 Open the context menu in the **Project Explorer** for the NaturalONE project into which you want to generate the code frame.

   Or:

   Open the context menu in the **Project Explorer** for the **Construct** root node or **Construct > Code-Frames** node into which you want to generate the code frame.

2 Select **Code Generation > New Code Frame**.

The **Define code frame details** panel is displayed. For example:



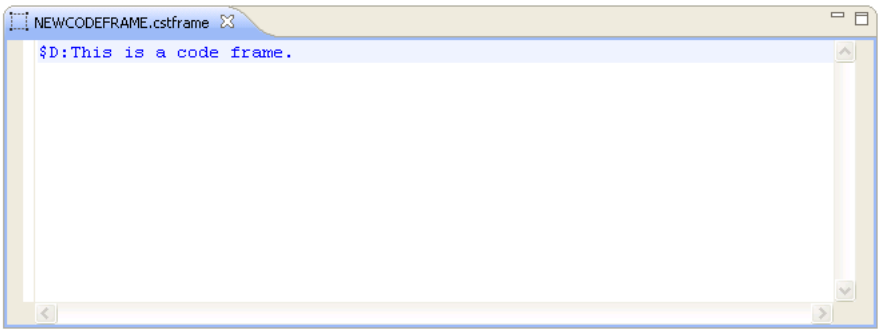
Note: To change the name of the project, type the name of a new project in **Project** or select **Browse** to display the available projects for selection.

3 Type the name of the new code frame in **Name**.

4 Type a brief description of the code frame in **Description**.

5 Select **Finish**.

The new code frame is displayed in the editor. For example:



6 Define the new code frame.

For information, see *Use the Code Frame Editor*.

7 Save the specifications for the code frame.

You can now upload the new code frame to your Natural Construct installation on the server.

Modify an Existing Code Frame

This section describes how to modify a Natural Construct code frame in the local environment.

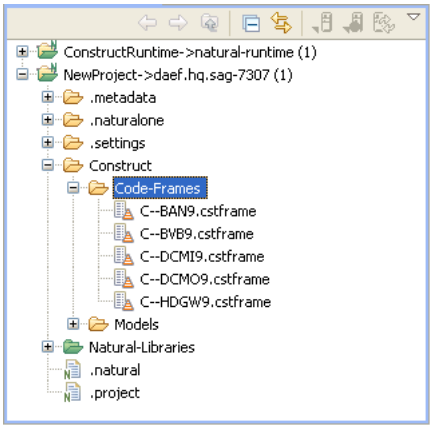
To modify an existing code frame

1 Download the code frame from the **Natural Server** view.

For information, see *Download Natural Construct Resources to a Local Project*.

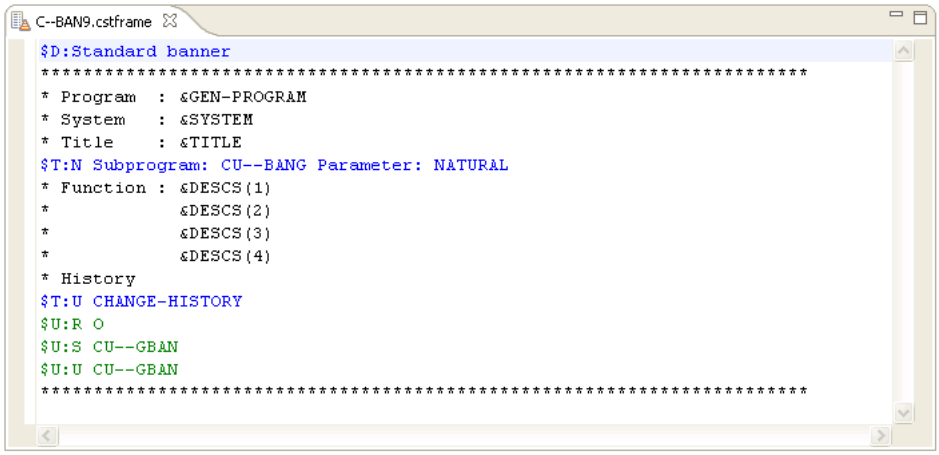
2 Expand the **Construct > Code-Frames** root node in the **Project Explorer**.

For example:



3 Open the code frame file (.cstframe extension) in the editor.

For example:



4 Modify the code frame as desired.

For information about the editor, see *Introduction*.

5 Save the code frame changes.

6 Upload the code frame to the Natural Construct server installation using standard NaturalONE functionality.

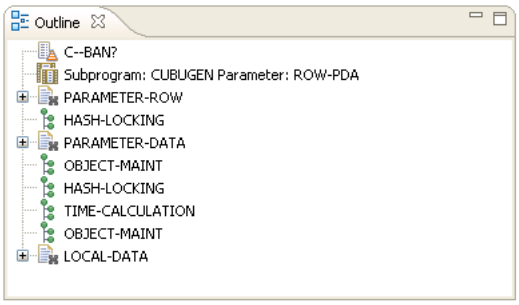
View a Code Frame in the Outline View

When editing a code frame in the code frame editor, the **Outline** view displays the main code frame editor statements (Condition code lines, Type code lines, User exit lines, etc.) in a tree form, using the condition codes to determine the parent/child relationships.

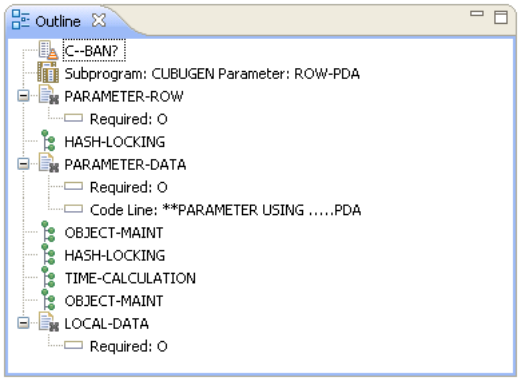
The following example illustrates the CBUA9 code frame in the code frame editor:

```
CBUA9.cstframe
$D:Object Browse Select Subp define data area
$T:F C--BAN?
DEFINE DATA
*
* Object Browse Select data
PARAMETER                        /* Rows
$T:N Subprogram: CUBUGEN Parameter: ROW-PDA
$T:U PARAMETER-ROW
$U:R O
    03 EXTRA-ROW-DATA
      04 ROW-STATE(A2)
$C:1 HASH-LOCKING
$C:"       04 ROW-HASH (B20)
      04 ROW-ID (N5) /* for internal use; do not change
      04 ROW-ERROR-DATA
        05 ##ERROR-FIELD (A32)
        05 ##MSG-NR (N4)
        05 ##MSG (A&LENGTH-OF-ROW-MESSAGE)
$T:U PARAMETER-DATA
$U:R O
$U:L **PARAMETER USING .....PDA
  PARAMETER USING CDBUPDA        /* Business service status data
  PARAMETER USING CDBUINFO       /* Business service message data
*
* Object Browse data
  PARAMETER USING &PRIVATE-PDA-NAME         /* Internal
  PARAMETER USING &KEY-PDA-NAME         /* Keys
*
  LOCAL USING &STATIC-OBJECT-PDA            /* Static object browse data
  LOCAL USING CDBRPDA        /* Standard
*
$C:1 OBJECT-MAINT
$C:" * Object Maint data
$C:"   LOCAL USING &OBJECT-PDA         /* Object
$C:"   LOCAL USING &RESTRICTED-PDA         /* Internal
$C:"   LOCAL USING CDAOBJ2        /* Standard
$C:1 HASH-LOCKING
$C:"   LOCAL USING CDHASHA        /* Hash locking data
$C:"   LOCAL USING &OBJECT-LDA    /* Hash locking LDA for hash calculations
```

The code frame is also displayed in the **Outline** view. For example:



Expand each node to display the data. For example:



When you select a node in the **Outline** view, the corresponding item is also highlighted in the code frame editor. For example, if you select the "Subprogram: CUBUGEN Parameter: ROW-PDA" node in the **Outline** view, the code frame editor will automatically highlight the "$C:" $T:N Subprogram: CUBUGEN Parameter: ROW-PDA" line. Conversely, when you select an item in the code frame editor, the corresponding node is highlighted in the **Outline** view.

The **Outline** view does not display all code frame lines in the editor. The following lines are displayed:

- Condition code statements that indicate different levels ($C:1-9).
- Type code statements that indicate subprograms ($T:N), code frames ($T:F), user exits ($T:U), and conditional user exits ($T:X).
- User exit property statements that indicate user exit required ($U:R), generate as subroutine ($U:G), sample subprogram ($U:S), GUI sample subprogram ($U:U), and code frame lines ($U:L).

Create and Maintain a Natural Construct Model UI

This section describes how to create and maintain the user interface (UI) for a Natural Construct model.

The following topics are covered:

- Create a New Model UI
- Create a New Dialog UI
- Create a New Page UI

Create a New Model UI

This section describes how to generate and maintain a model UI file.

The following topics are covered:

- Generate the Model UI File
- Copy a Model UI File
- Dependencies View
- Outline View
- Modify an Existing Model UI

Generate the Model UI File

To generate a new Natural Construct model UI file

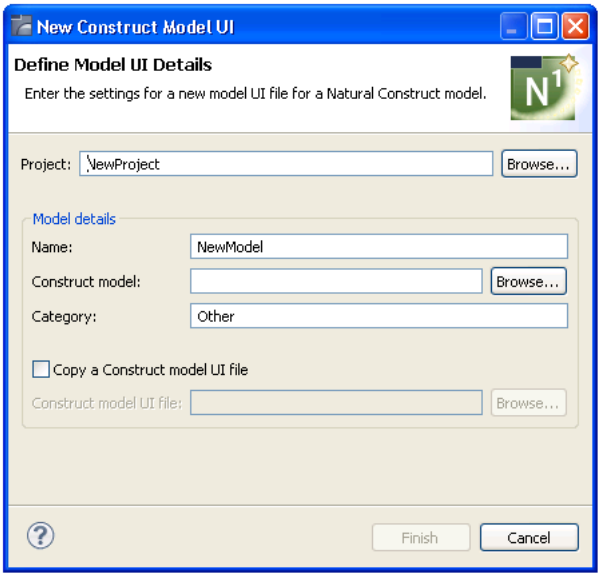
1 Open the context menu in the **Project Explorer** for the NaturalONE project into which you want to generate the model UI file.

Or:

Open the context menu in the **Project Explorer** for the **Construct** root node or **Construct > Models** node into which you want to generate the model UI file.

2 Select **Code Generation > New Construct Model UI**.

The **Define Model UI Details** panel is displayed. For example:



Note: To change the name of the project in which to generate the model UI, type the name of a new project in **Project** or select **Browse** to display the available projects for selection.

3 Type the name of the model UI file in **Name**.

4 Type the name of the Construct model file (.cstmdl extension) for which you are creating the interface model in **Construct model**.

Or:

Select **Browse**.

A selection window is displayed, listing the .cstmdl files for the standard models. Select the name of the Construct model file for which you are creating the interface and select **OK**. The file name is then displayed in **Construct model**.

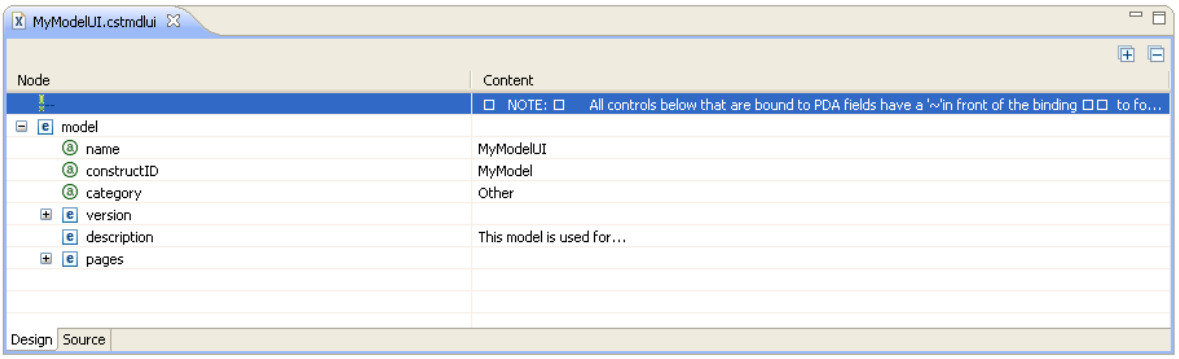
Note: Alternatively, you can copy a Natural Construct model UI file and modify it to suit your requirements. For information, see *Copy a Model UI File*.

5 Type the name of a category in **Category**.

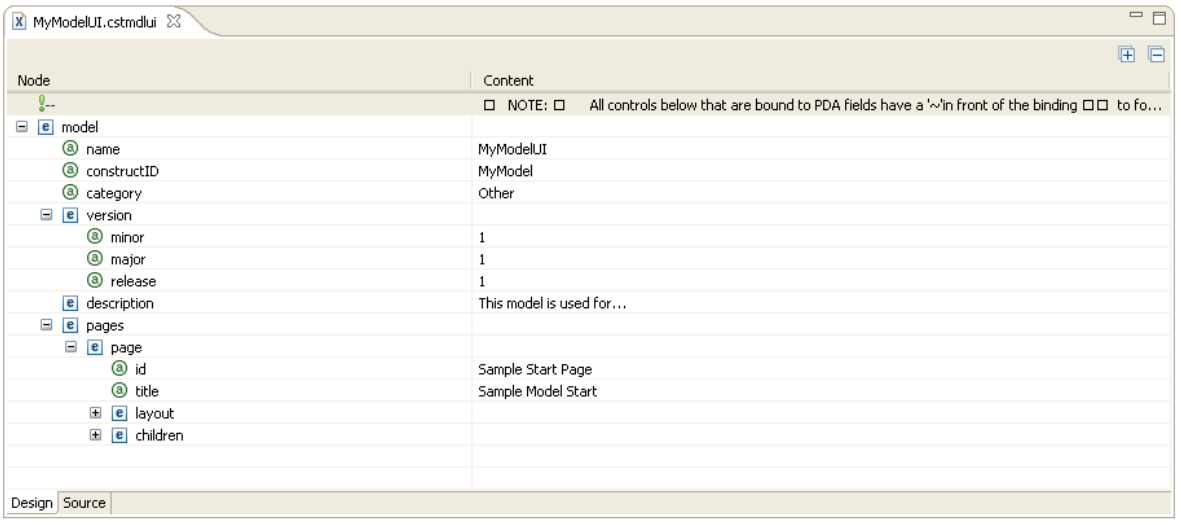
Categories are used to sort models for selection.

6 Select **Finish**.

The XML file for the model is generated and a simplified representation of the file is displayed in the editor. Each entry displayed in the **Design** tab corresponds to an entry in the XML file for the model. For example:



In this example, you can expand the **version** and **pages** nodes to view other nodes and contents:



Note: Any control listed in the **Design** tab that is bound to a PDA field has a "~" character preceding the binding, which forces the binding to be invalid when the field does not exist in the PDA. To enable the binding, add the field definition to the PDA and then delete the "~" character.

7 Select the **Source** tab.

The generated XML file (.cstmdlui extension) is displayed. For example:

```
MyModelUI.cstmdlui
<!--
   NOTE:
      All controls below that are bound to PDA fields have a '~'in front of the binding
      to force the binding to be invalid because the field may not exist in the PDA.
      To enable the binding  add the field definition to the PDA and then delete the '~' characters.

 01 FIELDS
   02 Description (A100/1:4)
   02 PDA (A) DYNAMIC
   02 logical (L)
   02 Date (D)
   02 Time (T)

-->
<model name="MyModelUI" constructID="MyModel" category="Other">
    <version minor="1" major="1" release="1" />
    <description>This model is used for...</description>
    <pages>
        <page id="Sample Start Page" title="Sample Model Start">
            <layout class="gridLayout" columns="3" />
            <children>
                <!-- Project label, text and button combination -->
                <label text="Project:" />
                <text id="ProjectTextText" text="{specs:project}">
                    <layoutData class="gridLayoutData" horizontalSpan="1"
                        grabExcessHorizontalSpace="true" />
                </text>
                <cstBrowseProject allowDefault="true" />
                <!-- Library label, text and button combination -->
                <label text="Library:" />
                <text id="LibraryText" text="{specs:library}">
                    <layoutData class="gridLayoutData" horizontalSpan="1"
                        grabExcessHorizontalSpace="true" />
                </text>
                <cstBrowseLibrary allowDefault="true"/>
                <label text="Module:" />
                <!-- Sample simple text binding -->
                <text id="ModuleText" text="~{specs:Module}">
Design  Source
```

The default settings used to generate the file are based on the Construct model selected on the **Define Model Details** panel.

8 Define the settings for the model UI.

For information, see *Create a New Client Generation Wizard*.

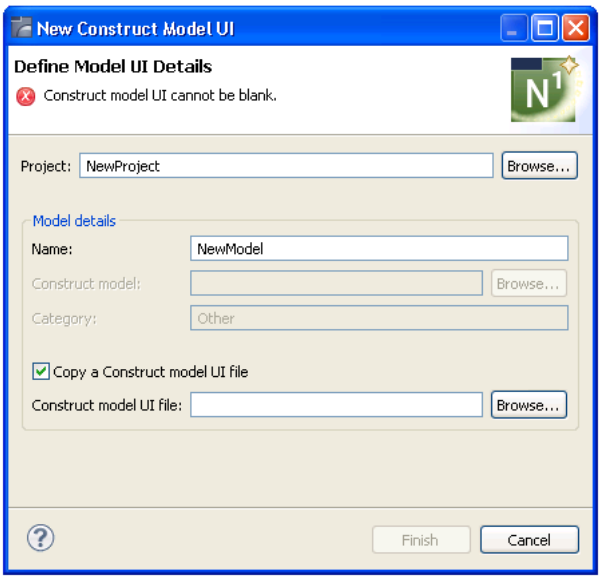
Copy a Model UI File

This section describes how to create a new model UI file from an existing model UI file supplied with NaturalONE and the Natural Construct plug-in. Using this method is a quick way to create your own model UI files by modifying existing files to suit your requirements. If the selected model UI file includes any reusable dialog or page UI files that do not currently exist in the workspace, these files will also be copied.

To copy a model UI file

1 Select **Copy a Construct model UI file**.

Construct model and **Category** are disabled and **Construct model UI file** is enabled. For example:



2 Type the path for a Natural Construct model UI file (.cstmdlui extension) in **Construct model UI file** (for example, C:\folder*filename*.cstmdlui).

Or:

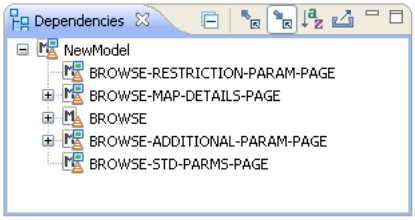
Select **Browse**.

A selection window is displayed, listing the .cstmdlui files for the supplied models. Select the file you want to copy and select **Open**. The location of the file is then displayed in **Construct model UI file**.

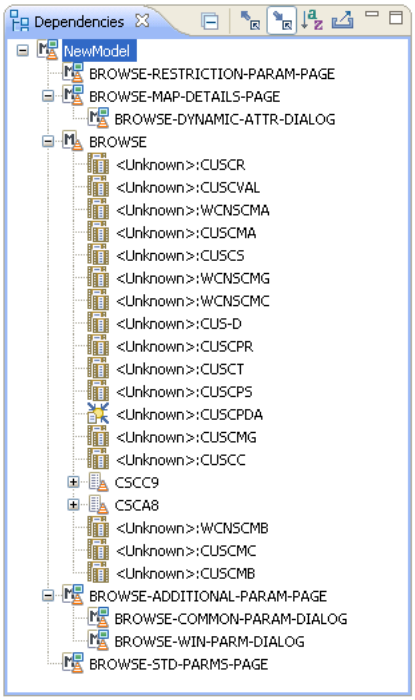
Note: When this option is not selected, **Construct model** and **Category** are enabled and must be specified.

Dependencies View

This view lists all modules referenced by the model UI file you are creating, including the modules shipped with the Construct runtime project. For example:



In this example, NewModel was created by copying the Natural Construct Browse model UI file and the **Dependencies** view displays model UI file, as well as the reusable pages used by that model. Expand the nodes to view the dependencies. For example:



If <Unknown> is displayed beside the name of a referenced module, the module is not available within the current project or referenced locally. You must either create the module locally or download it from the server. Any required compile/runtime modules are shipped in the Construct runtime project.

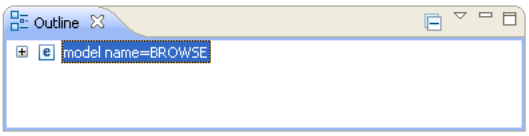
Notes:

1. For more information about the **Dependencies** view, see the description of the source editor in *Using NaturalONE*.
2. To reference modules in a local project, use the **Properties** window for the current project.
3. To download modules from the server, see *Download Natural Construct Resources to a Local Project*.

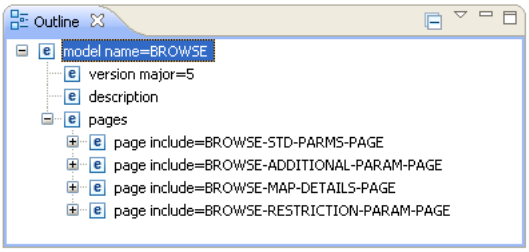
4. To add the compile/runtime modules in the Construct runtime project, see *Add the Construct Runtime Project*.

Outline View

The **Outline** view displays an outline of the settings defined in the **Design** tab. For example:



Expand the model name node. For example:



The model UI file in this example was copied from the BROWSE model UI file, which included several reusable pages.

Note: For information about reusable pages, see *Reusable Dialog and Page UI Files*.

Modify an Existing Model UI

This section describes how to modify an existing model UI file.

To modify an existing model UI file

1 Open the model UI file (.cstmdlui extension) in the editor.

2 Modify the model UI information as desired.

For information, see *Create a New Client Generation Wizard*.

3 Save the model UI changes.

Create a New Dialog UI

This section describes how to create and maintain the user interface (UI) for a Natural Construct dialog UI file, a reusable file that can be included in multiple model UI files.

The following topics are covered:

- Generate a Dialog UI File
- Modify an Existing Dialog UI

Note: For more information, see *Reusable Dialog and Page UI Files*.

Generate a Dialog UI File

To generate a new dialog UI file

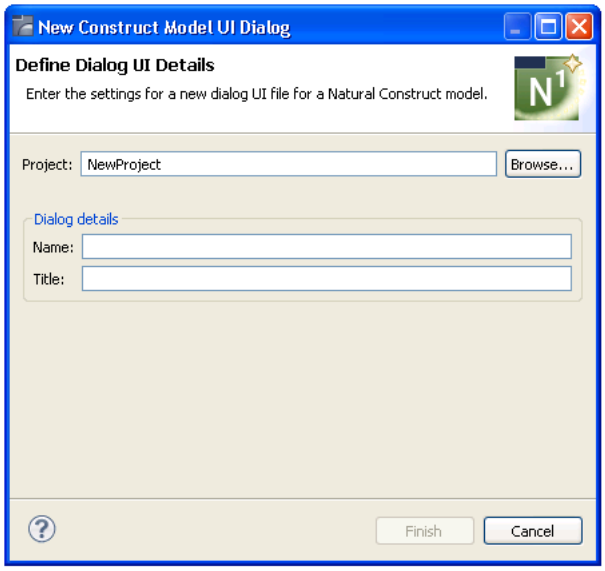
1 Open the context menu in the **Project Explorer** for the NaturalONE project into which you want to generate the dialog UI file.

Or:

Open the context menu in the **Project Explorer** for the **Construct** root node or **Construct > Models** node into which you want to generate the dialog UI file.

2 Select **Code Generation > New Construct Model UI Dialog**.

The **Define Dialog UI Details** panel is displayed. For example:



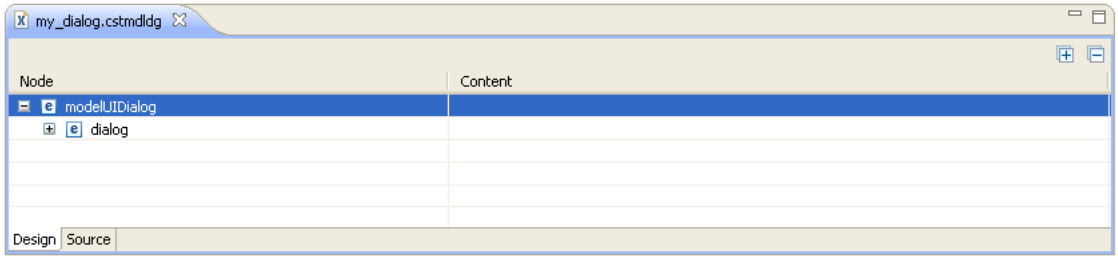
Note: To change the name of the project in which to generate the dialog UI file, type the name of a new project in **Project** or select **Browse** to display the available projects for selection.

3 Type the name of the new dialog UI file in **Name**.

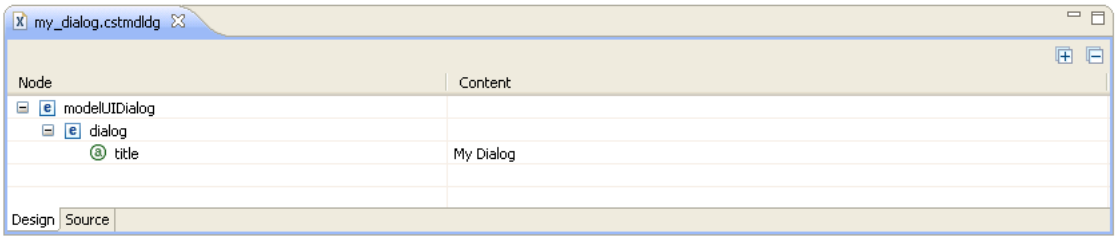
4 Type a title for the dialog in **Title**.

5 Select **Finish**.

The XML file (.cstmdldg extension) for the reusable dialog is generated and a simplified representation of the file is displayed in the editor in the **Design** tab. Each entry corresponds to an entry in the .cstmdldg file. For example:



In this example, you can expand the **dialog** node to view the contents:



Note: Any control listed in the **Design** tab that is bound to a PDA field has a "~" character preceding the binding, which forces the binding to be invalid when the field does not exist in the PDA. To enable the binding, add the field definition to the PDA and then delete the "~" character.

6 Select the **Source** tab.

The generated skeleton file is displayed. For example:

7 Define the settings for the dialog UI.

For information, see *Dialog Node*.

Modify an Existing Dialog UI

This section describes how to modify an existing dialog UI file.

To modify an existing dialog UI file

1 Open the dialog UI file (.cstmdldg extension) in the editor.

2 Modify the dialog UI information as desired.

For information, see *Dialog Node*.

3 Save the dialog UI changes.

Create a New Page UI

This section describes how to create and maintain the user interface (UI) for a Natural Construct page UI file, a reusable file that can be included in multiple model UI files.

The following topics are covered:

- Generate a Page UI File
- Modify an Existing Page UI

Note: For more information, see *Reusable Dialog and Page UI Files*.

Generate a Page UI File

To generate a new page UI file

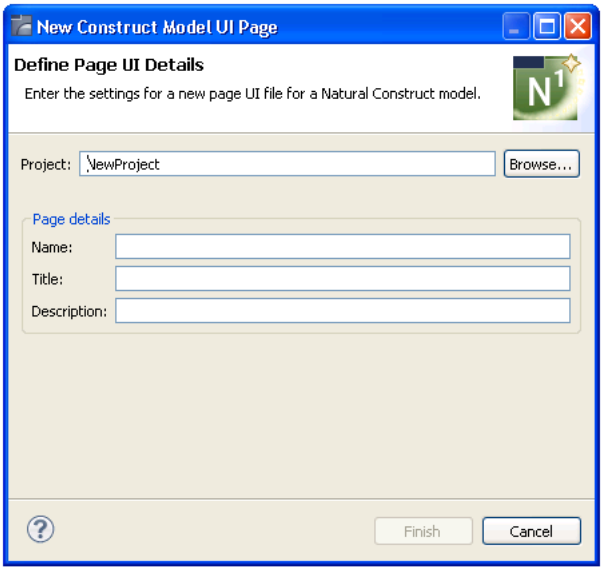
1 Open the context menu in the **Project Explorer** for the NaturalONE project into which you want to generate the page UI file.

Or:

Open the context menu in the **Project Explorer** for the **Construct** root node or **Construct > Models** node into which you want to generate the page UI file.

2 Select **Code Generation > New Construct Model UI Page**.

The **Define Page UI Details** panel is displayed. For example:



Note: To change the name of the project in which to generate the page UI file, type the name of a new project in **Project** or select **Browse** to display the available projects for selection.

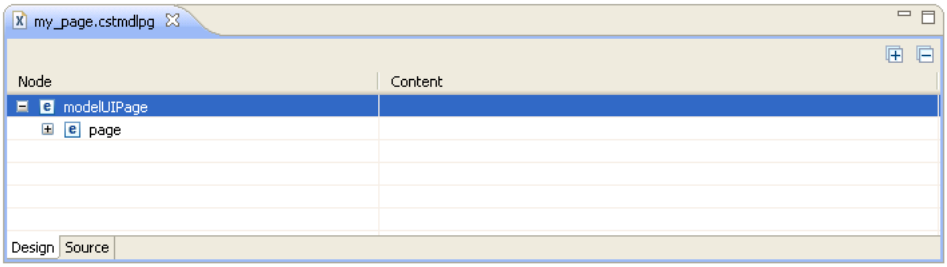
3 Type the name of the new page UI file in **Name**.

4 Type a title for the page in **Title**.

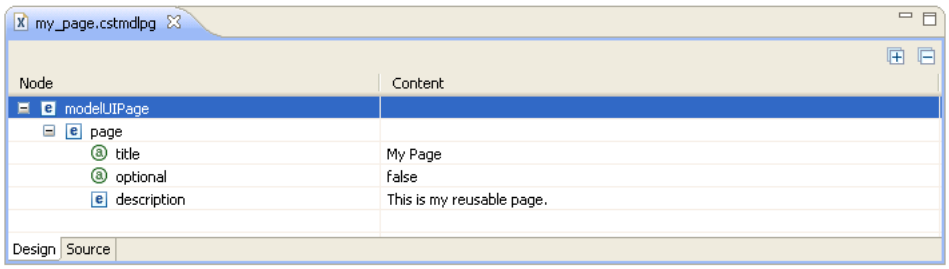
5 Type a description of the page in **Description**.

6 Select **Finish**.

The XML file (.cstmdlpg extension) for the reusable page is generated and a simplified representation of the file is displayed in the editor in the **Design** tab. Each entry corresponds to an entry in the .cstmdlpg file. For example:



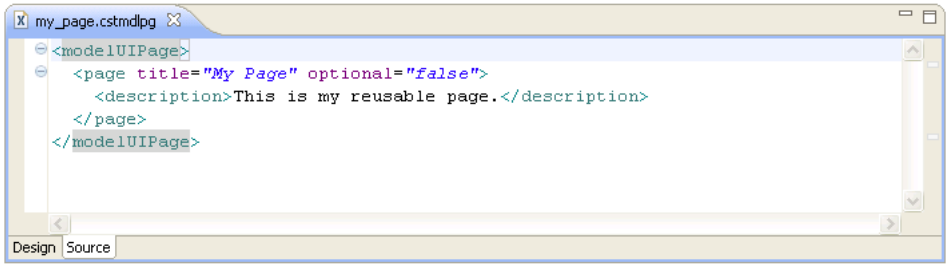
In this example, you can expand the **page** node to view the contents. For example:



Note: Any control listed in the **Design** tab that is bound to a PDA field has a "~" character preceding the binding, which forces the binding to be invalid when the field does not exist in the PDA. To enable the binding, add the field definition to the PDA and then delete the "~" character.

7 Select the **Source** tab.

The generated skeleton file is displayed. For example:



8 Define the settings for the page UI.

For information, see *Page Node*.

Modify an Existing Page UI

This section describes how to modify an existing page UI file.

To modify an existing page UI file

1 Open the page UI file (.cstmdlpg extension) in the editor.

2 Modify the page UI information as desired.

For information, see *Page Node*.

3 Save the page UI changes.

13 Set Natural Construct Preferences

- Set Construct Preferences 272
- Set Installation Preferences 274

Set Construct Preferences

This section describes how to set preferences for Natural Construct resources.

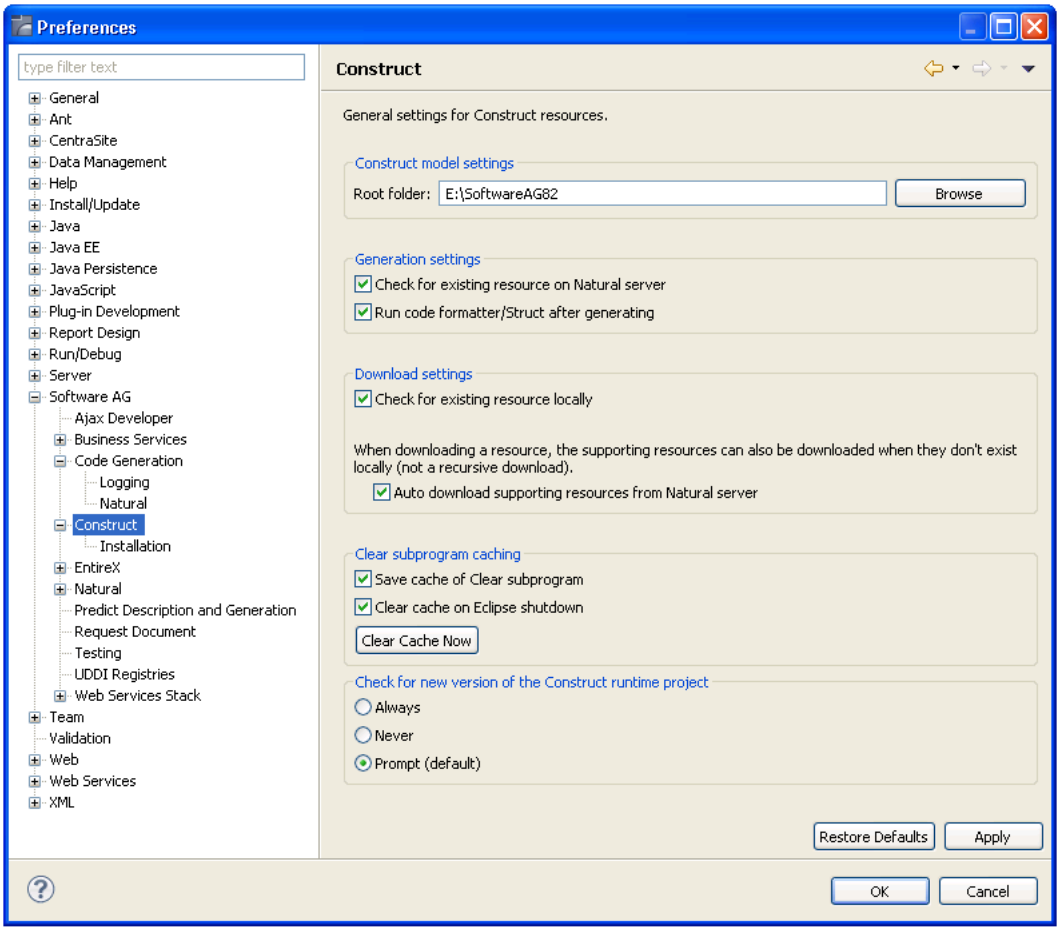
To set Construct preferences

1 Select **Preferences** on the **Window** menu.

The **Preferences** window is displayed.

2 Select **Software AG > Construct**.

The **Construct** preferences are displayed. For example:



Using this window, you can:

Task	Procedure
Select the root folder to use for models.	Type or select the root folder in **Root folder**.
Disable the search for existing resources on the Natural server while generating new Construct resources.	Deselect **Check for existing resource on Natural server**. By default, this preference is selected and the generation of a new Construct resource will initiate a search for the resource on the Natural server. If a resource exists on the server, a warning is displayed and the user can either generate the resource locally by selecting **OK** or cancel the generation of the resource by selecting **Cancel**.
Disable the Struct functionality to format code after generating (and regenerating) using Construct.	Deselect **Run code formatter/Struct after generating** and Construct-generated code will contain the original indentation indicated in the code frames. By default, this preference is selected and all Construct-generated code is formatted using the Struct option.
Disable the search for existing resources locally while downloading resources from the server.	Deselect **Check for existing resource locally**. By default, this preference is selected and the download of Construct resources from the server will initiate a search for the resources locally. If a resource exists locally, a warning is displayed and the user can either continue the download process by selecting **Yes** or **Yes to All**, which will overwrite the local copy, or suppress the download by selecting **No**, **No to All**, or **Cancel**.
Disable the automatic download of supporting resources from the Natural server when they do not exist locally.	Deselect **Auto download supporting resources from Natural server**. By default, this preference is selected and the download of Construct resources from the server will initiate a search for all supporting resources locally. If a supporting resource (i.e., a code frame or model) does not exist locally, it will also be downloaded. **Note:** This setting does not apply to recursive download operations.
Change options for the results of the server call by the clear subprogram when starting a client generation wizard.	By default, the cache of the clear subprogram is saved (**Save cache of clear subprogram**) and will be cleared when Eclipse is shut down (**Clear cache on Eclipse shutdown**). This allows the wizard to start faster on subsequent calls to the same clear subprogram. To disable this functionality, deselect the options in **Clear subprogram caching**.
Clear the cache of the clear subprogram immediately.	Select **Clear Cache Now**.
Change whether the version information for the Construct	By default, a prompt is displayed during startup, generation or regeneration, asking whether you want to update the Construct

Task	Procedure
runtime project is checked or not and when it is checked.	runtime project if a newer version is available. Other options are to: ■ Always update when a newer version is available, select **Always**. ■ Never prompt or automatically update the project when a newer version is available, select **Never**.

3 Select **OK** to save the preferences.

Set Installation Preferences

To function properly, certain UI functions require a Natural Construct installation on the Natural server. For example, the Construct root node in the **Natural Server** view can be used to download Natural Construct resources from a Natural server to a local Natural project, but only when there is a Natural Construct installation on the server. By default, these UI functions will be made visible based on the installation of Natural Construct on the Natural server. To accomplish this, a server call determines which products are installed on the server and the results are cached until Designer shuts down, which allows for only one server call per host|port|session parameter. An option is also provided to make these server calls and cache the results upon Designer startup (a server call for each mapped server in the workspace).

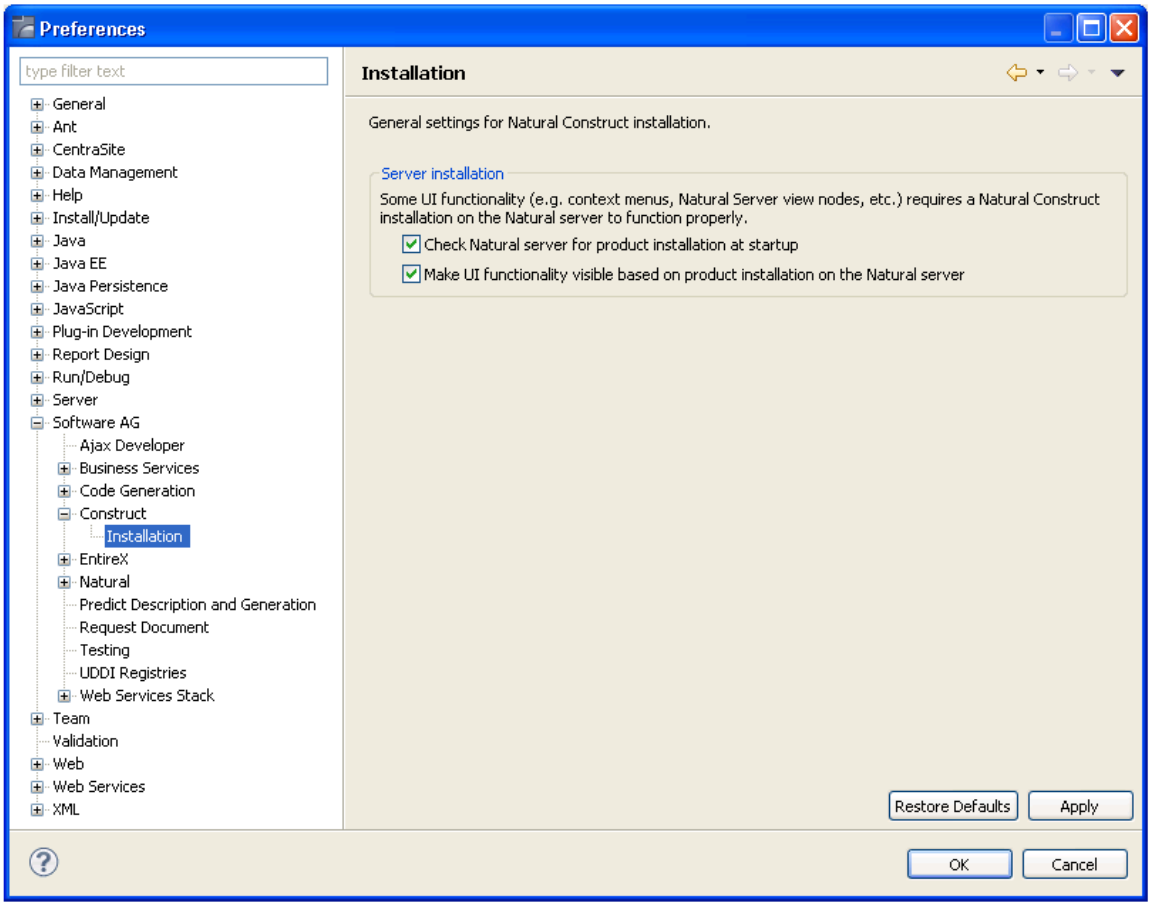
To set installation preferences

1 Select **Preferences** on the **Window** menu.

The **Preferences** window is displayed.

2 Select **Software AG > Construct > Installation**.

The **Preferences** window for Installation options is displayed. For example:



Using this window, you can:

Task	Procedure
Delay the server call to determine product installation until required (just prior to UI function visibility).	Deselect **Check Natural server product installation at startup**.
Make all UI functions visible, even when Natural Construct is not installed on the Natural server.	Deselect **Make UI functionality visible based on Natural server installation**. No server calls will be made to determine which products are installed on the server.

3 Select **OK** to save the preferences.

III

- 14 Defining User Exits 279
- 15 Using the Construct Runtime/Compile Time Modules in Non-Construct Server Environments 293
- 16 Generating an Ajax Page for Generated Subprograms 301

14 Defining User Exits

- Introduction 280
- Define a User Exit 280

This section explains what a user exit is and how to select and define one in NaturalONE.

Introduction

By default, the generated source code is protected from editing and changes can only be made within user exits, positions within the generated code where you can insert customized or specialized processing. Changes to the user exit code are always preserved upon subsequent regeneration of the module. We recommend that you only add custom code within user exits.

Caution: Although it is not recommended, you can edit the protected lines in the generated source code outside of the user exits. However, your changes will not be preserved upon regeneration. For information about the protected lines in the generated source code, see *Using the Source Editor/Protected Lines in Sources Generated by Construct or Code Generation* in *Using NaturalONE*.

The code generation wizards provide a wide variety of user exits, which vary based on the type of module you are generating. Some exits contain sample code or subprograms, while others generate the `**SAG DEFINE EXIT` and `**SAG END-EXIT` tags only — you provide the actual code. You can modify any user exit code generated into the edit buffer.

Notes:

1. For information on the supplied user exits, refer to *User Exits for the Generation Models*, *Natural Construct Generation*.
2. For information about the User Exit editor, refer to *User Exit Editor*, *Natural Construct Generation*.
3. If you require code to be inserted in the generated module where no user exit currently exists, have your Natural Construct administrator recommend a suitable exit or add a new exit to the wizard.

Define a User Exit

When a generated module is open in the Eclipse editor, all available user exits for the module are displayed in the **Outline** view.

This section covers the following topics:

- Access a User Exit
- Add Code to a User Exit
- Generate Sample
- Generate User Exit
- Clear Exit

- Modify Code in a User Exit

Tip: If the user exits are not displayed in the **Outline** view, select on the toolbar.

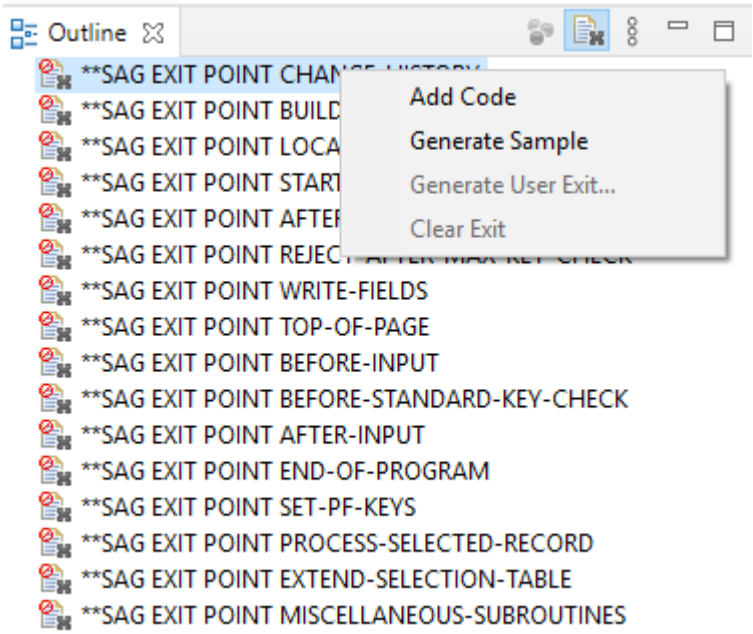
Access a User Exit

To access a user exit

1 Select the user exit in the **Outline** view.

2 Open the context menu for the user exit.

For example:



3 Select one of the options listed.

The user exit is displayed in the editor view. For example:

```
NEWBROW.NSN
* >Natural Source Header 000000
**SAG GENERATOR: BROWSE-SUBP                        VERSION: 5.3.1.12
**SAG TITLE: Browse ...
**SAG SYSTEM: NEWLIB
**SAG DESCS(1): This subprogram is used to browse the ...
**SAG HEADER1: First heading
**SAG DIRECT-COMMAND-PROCESS:
**SAG PRIME-FILE: NCST-WAREHOUSE
**SAG PFILE: NCST-WAREHOUSE
**SAG PRIME-KEY: WAREHOUSE-ID
**SAG MAX-WINDOWS: 01
**SAG MAX-PAGES: 10
**SAG DYNAMIC-ATTRIBUTES: ><
************************************************************************
* Program  : NEWBROW
* System   : NEWLIB
* Title    : Browse ...
* Generated: Oct  5,2011 at 19:45:39 by PWRUSR
* Function : This subprogram is used to browse the ...
*
*
*
* History
**SAG EXIT POINT CHANGE-HISTORY
************************************************************************
DEFINE DATA
PARAMETER
01 #PDA-KEY(A3)  /* Start/Returned key.
PARAMETER USING CDSELPDA /* Selection info
PARAMETER USING CDPDA-D  /* Dialog info
PARAMETER USING CDPDA-M  /* Msg info
PARAMETER USING CDPDA-P  /* Misc pass info
```

The `**SAG EXIT POINT` tag indicates that the exit does not exist and you must use the **Outline** view to add code or generate sample code.

Add Code to a User Exit

To add code to a user exit

1 Open the context menu for the user exit.

2 Select **Add Code**.

The exit is displayed in the editor view. For example:

```
*NEWBROW.NSN
* >Natural Source Header 000000
**SAG GENERATOR: BROWSE-SUBP                    VERSION: 5.3.1.12
**SAG TITLE: Browse ...
**SAG SYSTEM: NEWLIB
**SAG DESCS(1): This subprogram is used to browse the ...
**SAG HEADER1: First heading
**SAG DIRECT-COMMAND-PROCESS:
**SAG PRIME-FILE: NCST-WAREHOUSE
**SAG PFILE: NCST-WAREHOUSE
**SAG PRIME-KEY: WAREHOUSE-ID
**SAG MAX-WINDOWS: 01
**SAG MAX-PAGES: 10
**SAG DYNAMIC-ATTRIBUTES: ><
************************************************************************
* Program  : NEWBROW
* System   : NEWLIB
* Title    : Browse ...
* Generated: Oct  5,2011 at 19:45:39 by PWRUSR
* Function : This subprogram is used to browse the ...
*
*
*
* History
**SAG DEFINE EXIT CHANGE-HISTORY
**SAG END-EXIT
************************************************************************
DEFINE DATA
PARAMETER
01 #PDA-KEY(A3)  /* Start/Returned key.
PARAMETER USING CDSELPDA /* Selection info
PARAMETER USING CDPDA-D  /* Dialog info
PARAMETER USING CDPDA-M  /* Msg info
```

The `**SAG DEFINE EXIT` and `**SAG END-EXIT` tags indicate that the user exit exists, even when there is currently no code in it, and you can define or modify the exit directly in the editor.

3 Move the cursor to the end of the `**SAG DEFINE EXIT` *`user-exit-name`* line.

4 Select Enter.

5 Add the code on the lines provided.

Generate Sample

To add code to a user exit

1 Open the context menu for the user exit.

2 Select **Generate Sample**.

The `**SAG DEFINE EXIT` and `**SAG END-EXIT` lines are displayed with sample code. For example:

```
*NEWBROW.NSN
* >Natural Source Header 000000
**SAG GENERATOR: BROWSE-SUBP                    VERSION: 5.3.1.12
**SAG TITLE: Browse ...
**SAG SYSTEM: NEWLIB
**SAG DESCS(1): This subprogram is used to browse the ...
**SAG HEADER1: First heading
**SAG DIRECT-COMMAND-PROCESS:
**SAG PRIME-FILE: NCST-WAREHOUSE
**SAG PFILE: NCST-WAREHOUSE
**SAG PRIME-KEY: WAREHOUSE-ID
**SAG MAX-WINDOWS: 01
**SAG MAX-PAGES: 10
**SAG DYNAMIC-ATTRIBUTES: ><
************************************************************************
* Program  : NEWBROW
* System   : NEWLIB
* Title    : Browse ...
* Generated: Oct  5,2011 at 19:45:39 by PWRUSR
* Function : This subprogram is used to browse the ...
*
*
*
* History
**SAG DEFINE EXIT CHANGE-HISTORY
* Changed on Dec  5,2011 by PWRUSR for release ____
* >
* >
* >
**SAG END-EXIT
************************************************************************
DEFINE DATA
PARAMETER
```

3 Move the cursor to the end of the `**SAG DEFINE EXIT` *`user-exit-name`* line.

4 Modify the sample code as required.

Generate User Exit

This context menu is enabled only for special user exits such as WRITE-FIELDS and LOCAL-DATA. If you select this option, a dialog wizard that collects data for generating a user exit code is displayed.

To use this functionality you need to install Natural Business Services version 8.2.2.6 or higher on the NDV server.

- Generate a LOCAL-DATA User Exit

- Generate a WRITE-FIELDS User Exit

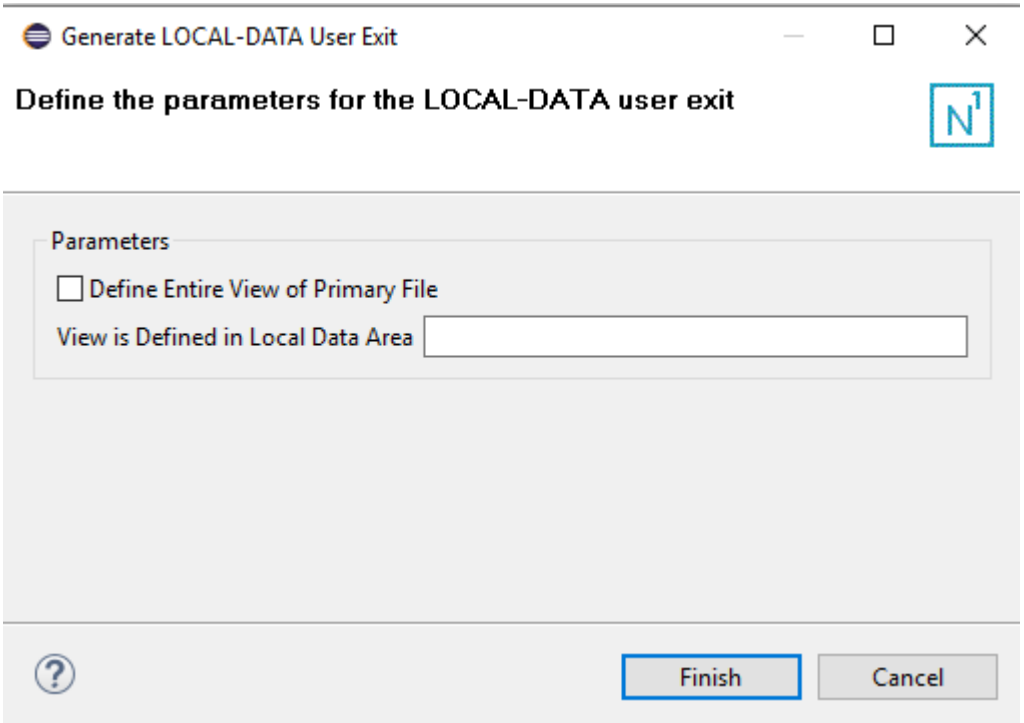
Generate a LOCAL-DATA User Exit

If you specify a program view name on a Browse, Browse-Select, or Object-Browse model specification wizard panel, you must define the view for the file as a LOCAL-DATA user exit.

To generate a Local-Data user exit

1 Open the context menu for the user exit.

2 Select **Generate User Exit**. The **Generate LOCAL-DATA** user exit dialog wizard is displayed.



Using this window, you can:

Field	Description
Define Entire View of Primary File	Mark this field to define the entire view of a primary file within a user exit. You can then edit the sample code and delete the fields you do not want.
View is Defined in Local Data Area	If the view is defined in a local data area (LDA), enter the name of the LDA in this field.

3 Modify this wizard as desired.

4 Click **Finish** to generate the user exit contents.

Note: If you are using the LOCAL-DATA user exit with a model that is not a Browse, Browse-Select, or Object-Browse model, no wizard is displayed.

Generate a WRITE-FIELDS User Exit

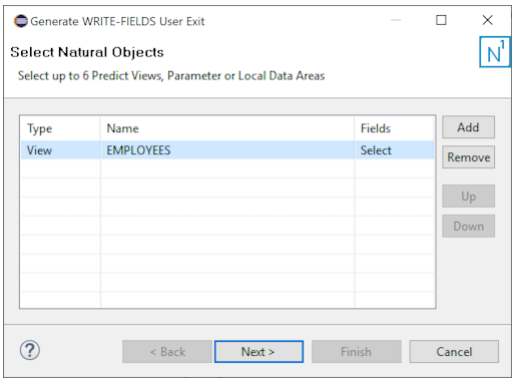
The WRITE-FIELDS user exit offers the possibility to generate more complex user exit code based on the contents of the #PANEL variable.

To generate a Write-Fields user exit

1 Open the context menu for the user exit.

2 Select **Generate User Exit**. The **Generate WRITE-FIELDS** user exit dialog wizard is displayed.

3 On the **Select Natural Objects** page, you can select up to six Predict views, local or parameter data areas. The Predict view from the model is added by default.



The following fields and buttons are available on this wizard page:

Field/Button	Description
Type	Shows the object type (View, Parameter or Local).
Name	Shows the name of the object.
Fields	Choose **Select** to display a selection page from which you can select the fields or **All** to select all the fields from the corresponding view.
Add	Shows a list of all available Predict views, local or parameter data areas and adds the objects you have selected. **Note:** The local and parameter data areas are retrieved in the library search order of the current project. **Note:** You may select several objects from one or different object types at once.
Remove	Removes the selected objects. **Note:** You may select several objects at once.

Field/Button	Description
Up	Moves the selected view up.
Down	Moves the selected view down.

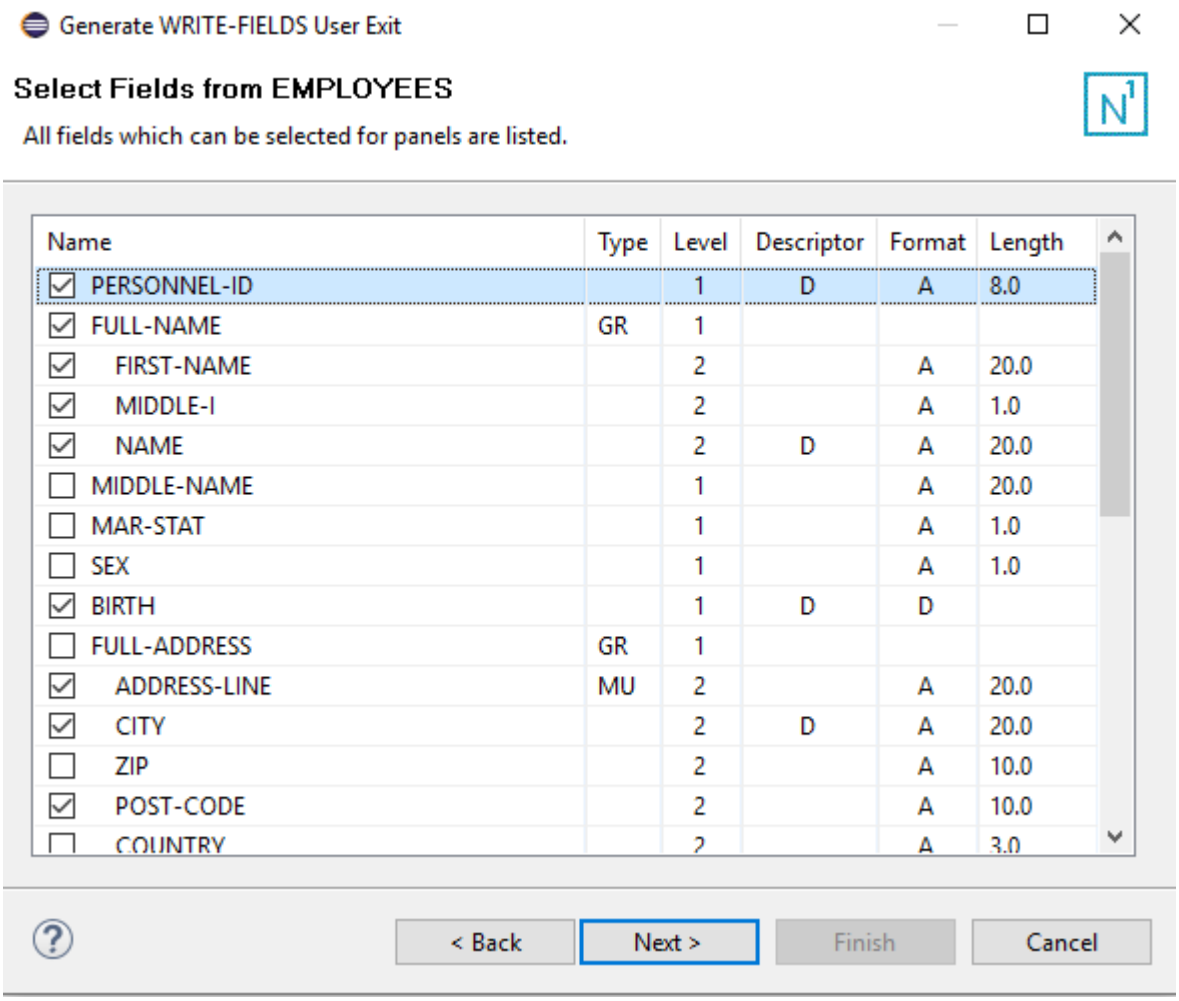
4 Modify this wizard as desired.

5 Click **Next >**.

6 The **Select Fields Page** is displayed. It shows all fields from all Predict views whose Fields parameter you have set to **Select**.

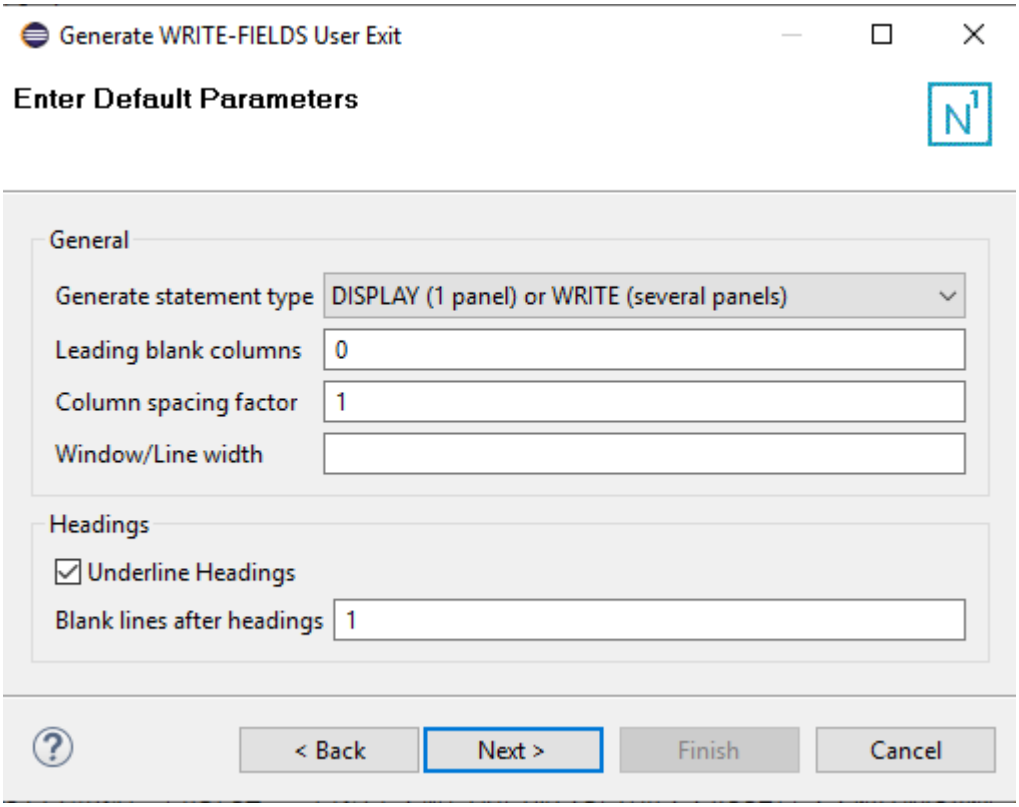
Select the fields to be used for the WRITE-FIELDS code.

If you select a group field, all children of the group will be selected.



7 Click **Next >**.

8 The **Enter Default Parameters** page is displayed. On this wizard page you can specify the statement type, size, and spacing of the report.



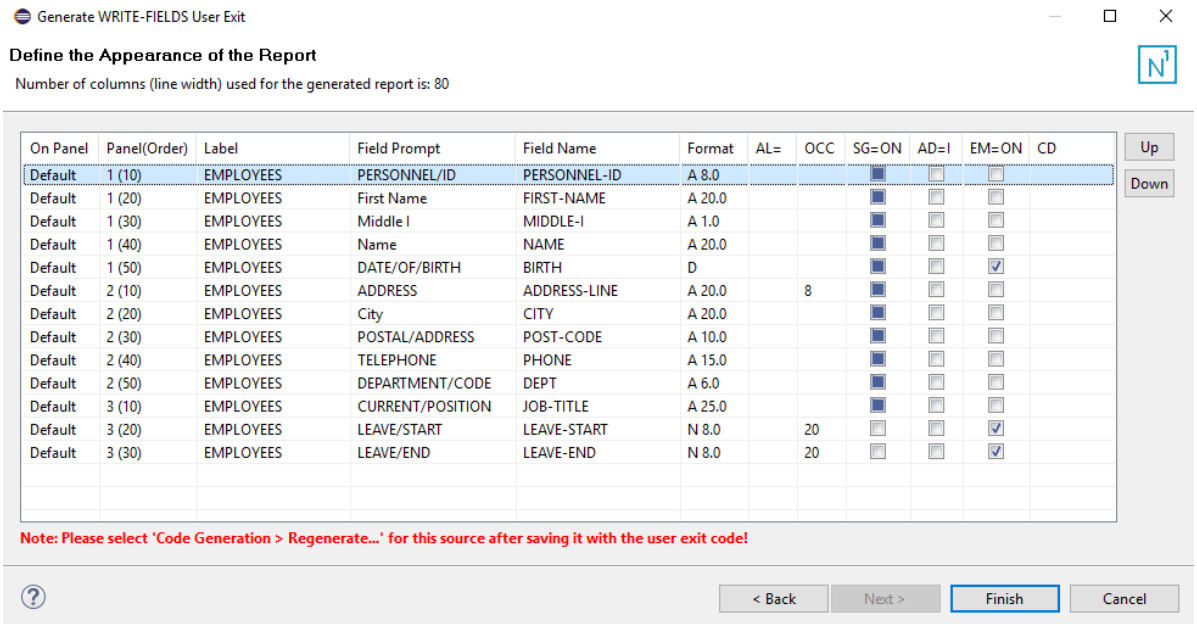
The following fields and buttons are available on this wizard page:

Field/Button	Description
Generate statement type	Type of Natural statement (DISPLAY or WRITE) used for the report. By default, DISPLAY is selected. If more than one panel is generated, but you have selected DISPLAY, the code will, nevertheless, be created using WRITE statements.
Leading blank columns	Number of blank columns preceding the report columns.
Column spacing factor	Number of spaces between fields on the report. The default is one space.
Window/Line width	Number of columns used for the generated report. If this field is empty, 80 is used.
Underline Headings	Tick this box to add or remove a line under headings. This field is selected by default.
Blank lines after headings	Number of blank lines below the header to be displayed in the ready report.

9 Modify this wizard as desired.

10 Click **Next >**.

11 The **Define the Appearance of the Report** page appears. On this wizard page you can define the layout of the WRITE-FIELDS panels.



On Panel	Panel(Order)	Label	Field Prompt	Field Name	Format	AL=	OCC	SG=ON	AD=I	EM=ON	CD
Default	1 (10)	EMPLOYEES	PERSONNEL/ID	PERSONNEL-ID	A 8.0			■	☐	☐	
Default	1 (20)	EMPLOYEES	First Name	FIRST-NAME	A 20.0			■	☐	☐	
Default	1 (30)	EMPLOYEES	Middle I	MIDDLE-I	A 1.0			■	☐	☐	
Default	1 (40)	EMPLOYEES	Name	NAME	A 20.0			■	☐	☐	
Default	1 (50)	EMPLOYEES	DATE/OF/BIRTH	BIRTH	D			■	☐	☑	
Default	2 (10)	EMPLOYEES	ADDRESS	ADDRESS-LINE	A 20.0		8	■	☐	☐	
Default	2 (20)	EMPLOYEES	City	CITY	A 20.0			■	☐	☐	
Default	2 (30)	EMPLOYEES	POSTAL/ADDRESS	POST-CODE	A 10.0			■	☐	☐	
Default	2 (40)	EMPLOYEES	TELEPHONE	PHONE	A 15.0			■	☐	☐	
Default	2 (50)	EMPLOYEES	DEPARTMENT/CODE	DEPT	A 6.0			■	☐	☐	
Default	3 (10)	EMPLOYEES	CURRENT/POSITION	JOB-TITLE	A 25.0			■	☐	☐	
Default	3 (20)	EMPLOYEES	LEAVE/START	LEAVE-START	N 8.0		20	☐	☐	☑	
Default	3 (30)	EMPLOYEES	LEAVE/END	LEAVE-END	N 8.0		20	☐	☐	☑	

The position of the fields is calculated automatically for any change affecting the field width, field order, or the line width of the report.

If new fields are selected by going back to the **Select Fields** wizard page, these fields are added at the end of the report.

The following fields and buttons are available on this wizard page:

Field/Button	Modifiable	Description
On Panel	x	Pick either **Default** or **All** from the drop-down list. **All** displays the field on all panels, and in such case the **Panel** value will be *. **Default** displays the field on one panel, and this case the **Panel** value will be the number of the panel where the field is located.
Panel(Order)		Number of the panel and order on this panel where the corresponding field will be displayed.
Label	x	Qualifier of the field. By default, the name of the view is selected.
Field Prompt	x	Field names displayed on the report. The default headings in Predict are displayed. Entering a slash (/) can split the header over several rows.
Field Name		The name of the field.
Format		The format and length of the field.
AL=	x	To override the default length of the corresponding field, type the new length in this field.

Field/Button	Modifiable	Description
OCC	depends on field	Number of occurrences of the corresponding field. If a fixed number is specified, that number is always displayed on the report (even if some occurrences are empty).
SG=ON	depends on field	Sign option. Check this box to include a sign option for the corresponding field (SG=ON) in the generated report.
AD=I	x	Check this box to intensify the corresponding field (AD=I).
EM=ON	depends on field	Check this box to use the Predict edit mask for the corresponding field (EM=ON).
CD	x	Select the color definition for the corresponding field from the drop-down list.
Up		Moves the selected field up.
Down		Moves the selected field down.

12 Click **Finish** to generate the user exit.

Depending on the input values, this generation could also create code in the user exits BUILD-REPORT-LOCAL-VARS and TOP-OF-PAGE.

Note: After creating the user exit code in the model source, you must save the source and regenerate it. Regeneration can be achieved by selecting the menu **Code Generation > Regenerate…** for the source.

Clear Exit

To clear code from an existing user exit

1 Open the context menu for the user exit in the **Outline** view.

2 Select **Clear Exit**.

All lines of code within the selected exit are deleted and the comment line that identifies the insertion point for the exit within the editor is restored.

Modify Code in a User Exit

To modify code in a user exit

1 Select the user exit in the **Outline** view.

The user exit is displayed in the editor.

2 Modify and save the user exit.

Note: You can make changes to user exits equivalent to the **Add Code** and **Clear Exit** options by modifying these lines in the editor view — without using the context menu for the **Outline** view. If you do this, ensure you do not change the ** SAG comment lines.

15 Using the Construct Runtime/Compile Time Modules in Non-Construct Server Environments

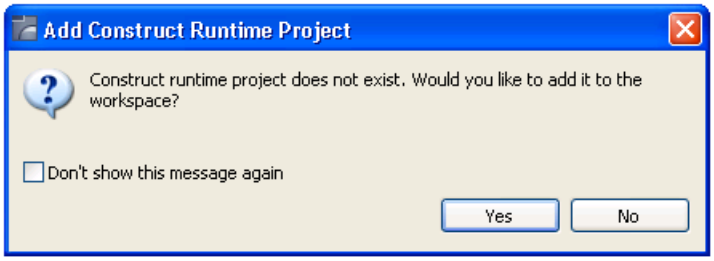
- Add the Construct Runtime Project 294
- Update the Construct Runtime Project to the Latest Version 295
- Replace the Construct Runtime Project with the Latest Version 296
- Exclude Modules from the Update or Replace Process 296
- Add Customized Modules to the Construct Runtime Project 297
- Build the Construct Runtime Project in a non-Construct Server Environment 298

To avoid compile errors for Construct-generated modules in the server environment, Natural Construct on the server is delivered with all the required runtime/compile time modules in the SYSTEM library. Since these modules are combined with other modules and products in SYSTEM, the Natural Construct component on the client delivers the required modules in the Construct runtime project. The modules in this project will eliminate compile and parsing errors caused by missing Natural Construct resources and will provide more detailed information in the **Dependencies** view. This project is available for use by both the client generation wizards and the Construct generation wizards.

If uploading runtime modules to the server causes compile or runtime errors to existing server modules, try regenerating the server modules to incorporate the changes in the uploaded runtime modules. Likewise, if the compilation or execution of generated code results in errors on the server, try rebuilding the Construct runtime project on the server to ensure that you are using the most recent version of this project.

Add the Construct Runtime Project

When Natural code is generated by a Code Generation wizard, the wizard verifies whether the required runtime/compile-time modules are available in the local environment. If they are not, a window is displayed prompting you to add the Construct runtime project to your workspace. For example:



If you select **Yes**, the project is added to the workspace and referenced from the current project.

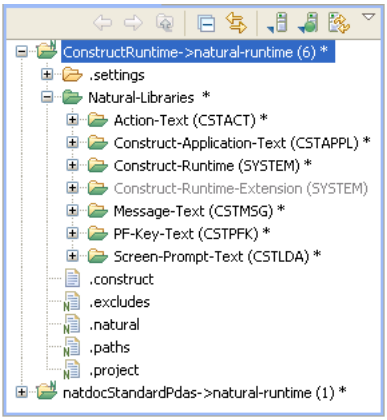
You can also add the Construct runtime project to your workspace manually.

To add the Construct runtime project manually

1 Open the context menu for any node in the **Project Explorer**.

2 Select **Code Generation > Construct Runtime > Add Runtime Project**.

The project is added to the **Project Explorer** view. For example:



The current version of the Construct runtime project is defined in the .construct file. For example:

Note: The information in the .construct file is used internally and should not be modified.

The local version information is compared to the version information delivered in the Construct runtime project at startup and during generation and regeneration. If the version has changed on the server, the local project will be updated.

Update the Construct Runtime Project to the Latest Version

This section describes how to update an existing Construct runtime project to the latest version of the project. Updated modules in the shipped version are copied to the workspace (and overwritten when necessary).

Caution: Any customizations of the Construct runtime project modules will be lost unless you exclude the modules from the update processing. For information, see *Exclude Modules from the Update or Replace Process*.

To update the Construct runtime project to the latest version

1 Open the context menu for the **ConstructRuntime** project in the **Project Explorer**.

2 Select **Code Generation > Construct Runtime > Update to Latest Version**.

Note: When the Construct runtime project is updated, the project modules are copied to the SYSTEM library on the FUSER, but when the project is built, the project modules are copied to the SYSTEM library on the FNAT.

Replace the Construct Runtime Project with the Latest Version

This section describes how to replace an existing Construct runtime project with the latest shipped version of the project. All modules in the shipped version are copied to the workspace (and overwritten when necessary).

Caution: Any customizations of the Construct runtime project modules will be lost unless you exclude the modules from the replace processing. For information, see *Exclude Modules from the Update or Replace Process*.

To replace the Construct runtime project with the latest version

1 Open the context menu for the **ConstructRuntime** project in the **Project Explorer**.

2 Select **Code Generation > Construct Runtime > Replace with Latest Version**.

Exclude Modules from the Update or Replace Process

This section describes how to exclude (and subsequently include) Construct runtime project resources from being overwritten during the update or replace process. You can use this functionality to protect changes to these modules from being overwritten during the update or replace process.

Notes:

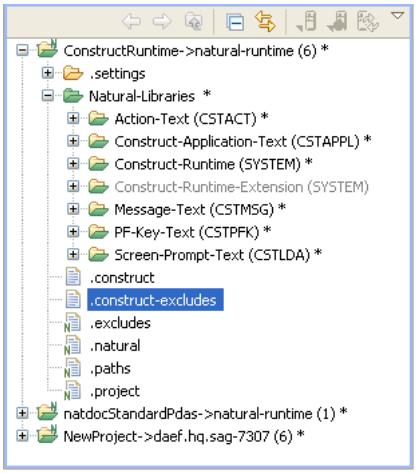
1. Excluding a folder automatically excludes all its child folders and files.
2. You cannot exclude the Construct runtime project itself or the .construct file.

To exclude modules in the Construct runtime project from the update or replace processing

1 Open the context menu for a Construct runtime resource in the **Project Explorer**.

2 Select **Code Generation > Construct Runtime > Exclude**.

The selected resource is added to the .construct-excludes file in the project. For example:



Excluded resources will never be updated or replaced with the latest shipped version (triggered automatically at startup, generation, regeneration or by selecting the **Update to Latest Version** or **Replace with Latest Version** context menu actions).

Note: You can view the .construct-excludes file to determine which resources are currently excluded, but you should never modify the file manually.

To include modules in the Construct runtime project update or replace processing

1 Open the context menu for the excluded Construct runtime resource in the **Project Explorer**.

2 Select **Code Generation > Construct Runtime > Include**.

The selected resource is removed from the .construct-excludes file and will now be overwritten during an update or replace process.

Add Customized Modules to the Construct Runtime Project

If you have customized any of the required modules on the server, you must add these customizations to the local Construct runtime project. This project is imported from an archived file called *ConstructRuntime.zip* in the installation folder for the Natural Construct component.

To add customized modules to the Construct runtime project

1 Make a backup copy of the *ConstructRuntime.zip* file.

2 Import the *ConstructRuntime.zip* file into your workspace.

3 Open the zip file and copy the customized modules into the Construct runtime project.

4 Export the modified Construct runtime project from your workspace to the *ConstructRuntime.zip* file.

   Your customizations will overwrite the supplied Construct runtime project in the Natural Construct installation folder.

The customized Construct runtime project now can be used as the basis for loading runtime projects in a customized environment.

Build the Construct Runtime Project in a non-Construct Server Environment

The Construct runtime project allows Construct-generated modules to be compiled in NaturalONE and executed in a non-Construct Natural server environment. If Natural Construct is installed on the server (including the compiled version only), a Construct-generated application can be compiled and/or executed on that server. If Natural Construct is not installed on the server (for example, in a NaturalONE local server environment), a Construct-generated application can be compiled on that server if the Construct runtime project has been installed.

The Construct runtime project is copied to the environment defined in the **Properties** window for the project (for example, `Projectname` > Properties > Natural > Runtime).

For a Construct-generated application to compile in a NaturalONE local environment:

- The referenced application DDMs must be copied into the NaturalONE local environment.
- The Construct runtime project must be available in the workspace and referenced.
- The SYSTEM library must be in the steplib chain.

The runtime folder does not require a Construct physical file. If Construct help is being used, the Construct physical file must be installed on the server and both the runtime and the CST-Help folders should be updated on the server.

Notes:

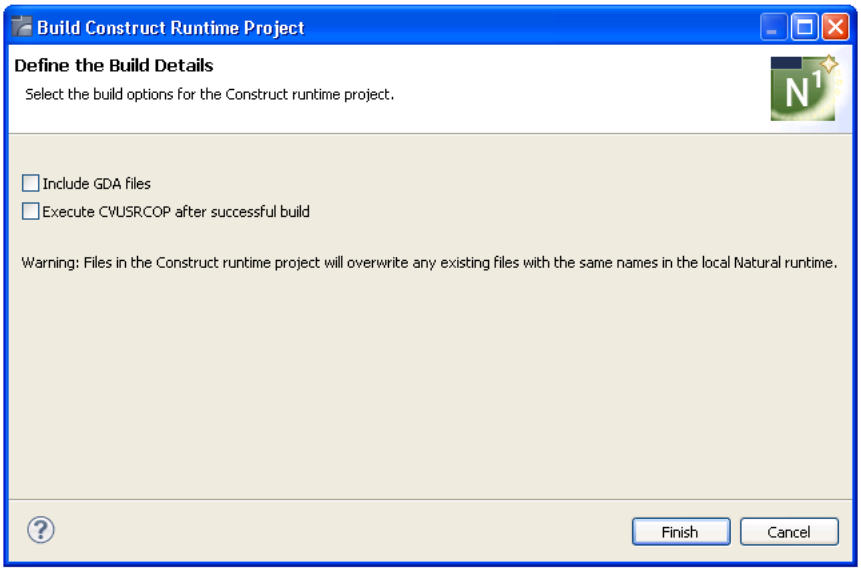
1. To build the Construct runtime project, it must first be available locally. For information, see ***Add the Construct Runtime Project***.
2. If your non-Construct server environment is running on a mainframe, ensure that the ESIZE value is a minimum of 120 during the build.
3. When the Construct runtime project is built, the project modules are copied to the SYSTEM library on the FNAT, but when the project is updated, the project modules are copied to the SYSTEM library on the FUSER.

To build the Construct runtime project in a non-Construct server environment

1 Open the context menu for the **ConstructRuntime** project in the **Project Explorer**.

2 Select **Code Generation > Construct Runtime > Build Construct Runtime Project**.

The **Define the Build Details** panel is displayed. For example:



3 Select **Finish**.

The Construct runtime project is built with all defaults.

Or:

Select one or more of the following options:

Option	Description
Include GDA files	Select this option to include all global data area files. The wizard ignores any global data areas. To include them, select Include GDA files. **Note:** To ensure compatibility with existing compiled files, and to avoid GDA timestamp errors, the Construct CDGDA global data area is also excluded by default. If you want this file uploaded and compiled, edit the .excludes file in the project. For more information on the .excludes file, see *Using NaturalONE*.
Execute CVUSRCOP after successful build	Select this option to execute the CVUSRCOP utility, which copies the Natural utility routines to the SYSTEM library after the Construct runtime project has

Option	Description
	been successfully built on the server. For more information, see *Natural Business Services Installation on Mainframes*.

4 Select **Finish**.

The Construct runtime project is built with the selected options.

16 Generating an Ajax Page for Generated Subprograms

- Generate an Ajax Page for an Object-Browse Subprogram 302
- Generate an Ajax Page for an Object-Maint Subprogram 310
- Generate an Ajax Main Program from an Adapter File 318
- Test the Generated Main Program 322
- Regenerate the Main Program 324

This section describes how to generate an Ajax page that takes advantage of the capabilities of a subprogram generated by an Object-Browse-Subp wizard (Object-Browse-N1 or Object-Browse-Subp) or an Object-Maint-Subp wizard (Object-Maint-N1 or Object-Maint-Subp). The following files are generated:

- page layout (.xml extension)
- adapter (.NS8 extension)
- main program (.NSP extension)

Generate an Ajax Page for an Object-Browse Subprogram

This section describes how to generate an Ajax page for an Object-Browse subprogram.

The following topics are covered:

- Access the Ajax Object-Browse Page Wizard
- Specify Source and Target Details for the Ajax Object-Browse Page Wizard
- Configure Column Details for the Ajax Object-Browse Page Wizard

Access the Ajax Object-Browse Page Wizard

The section describes how to access the Ajax Object-Browse Page wizard. Before accessing the wizard, the following conditions must be met:

- The wizard must be started from a project that has been enabled for Ajax Developer.
- A user interface component must be available locally.

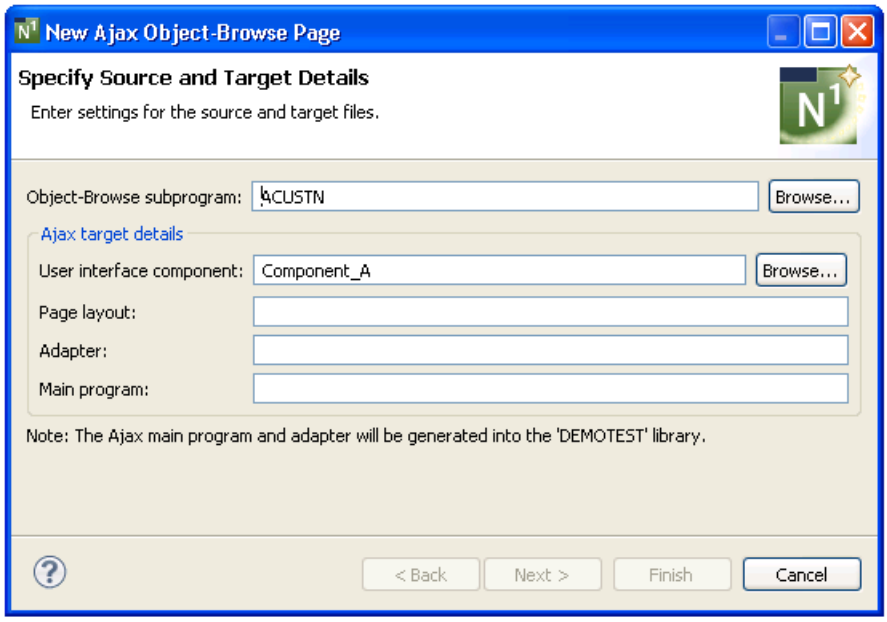
Note: For more information about the generated files and general Ajax architecture, refer to the Natural for Ajax documentation.

To access the Ajax Object-Browse Page wizard

1. Open the context menu in the **Project Explorer** for a subprogram that was generated by either the Object-Browse-N1 or Object-Browse-Subp generator.
2. Select **Code Generation > New Ajax Object-Browse Page**.

   For example:

   The **Specify Source and Target Details** panel is displayed. For example:



Specify Source and Target Details for the Ajax Object-Browse Page Wizard

To specify source and target details

1 Define the following parameters:

Parameter	Description
Object-Browse subprogram	Name of the Object-Browse-generated subprogram for which you are creating the page. To change the name of the subprogram, either type the name of a new subprogram in **Object-Browse subprogram** or select **Browse** to display the available subprograms for selection.
User interface component	Name of the user interface component for the page. Either type the name of an existing component in **User interface component** or select **Browse** to display the available components for selection.
Page layout	Name of the page layout file to be generated. This name must follow standard xml naming conventions (do not include the .xml extension). **Tip:** Avoid using spaces in the page name as it may cause problems during generation.
Adapter	Name of the adapter file to be generated. This name must follow standard Natural naming conventions (do not include the .NS8 extension).
Main program	Name of the main program file to be generated. This name must follow standard Natural naming conventions (do not include the .NSP extension).

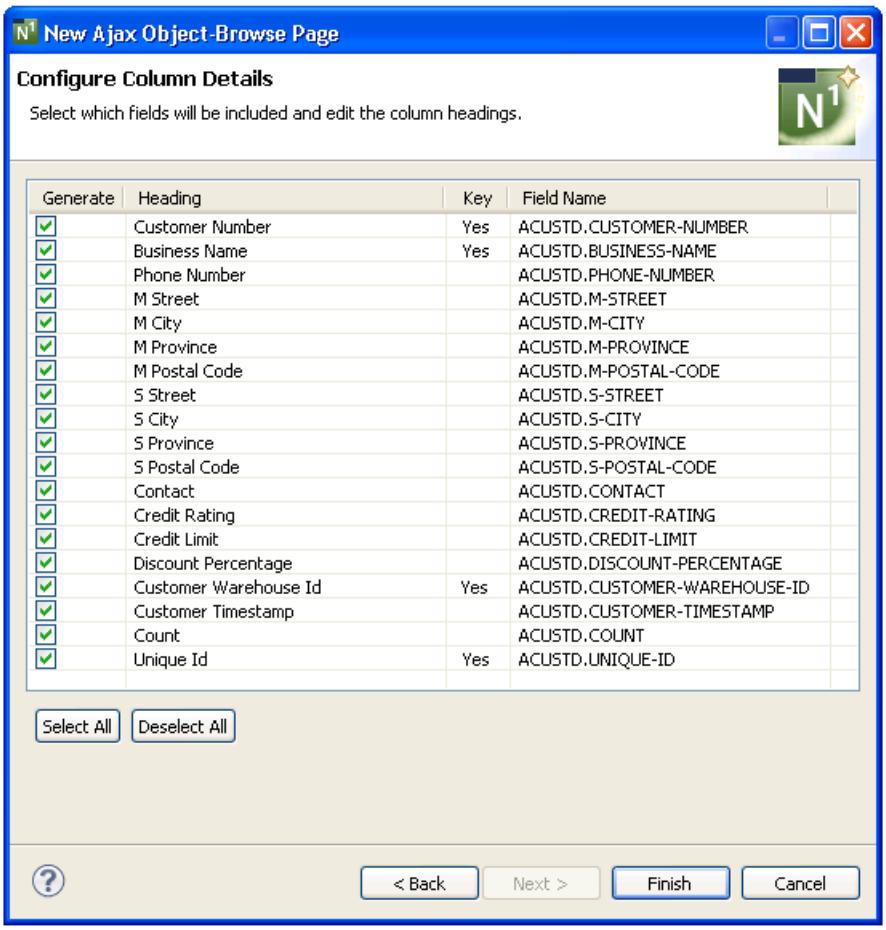
2 Select **Finish**.

The page is generated using all fields in the PDA, as well as all default column headings and search keys.

Or:

Select **Next**.

The **Configure Column Details** panel is displayed. For example:



Note: Array fields are not currently supported.

Configure Column Details for the Ajax Object-Browse Page Wizard

This panel allows you to select which fields are generated for the page and what column headings will be displayed. The **Key** and **Field Name** columns show parameters that are read-only. "Yes" in **Key** indicates that the corresponding field is used as a search key on the generated page; **Field Name** displays the fully qualified name of each field in the PDA.

To configure column details

1 Define the following parameters:

Parameter	Description	Required/Optional/Conditional
Generate	Indicates whether the corresponding field is generated or not. To exclude a field, deselect **Generate** for that field. A minimum of one field must be selected.	Optional
Heading	Heading displayed on the generated page for the corresponding field. You can change this heading as desired.	Optional

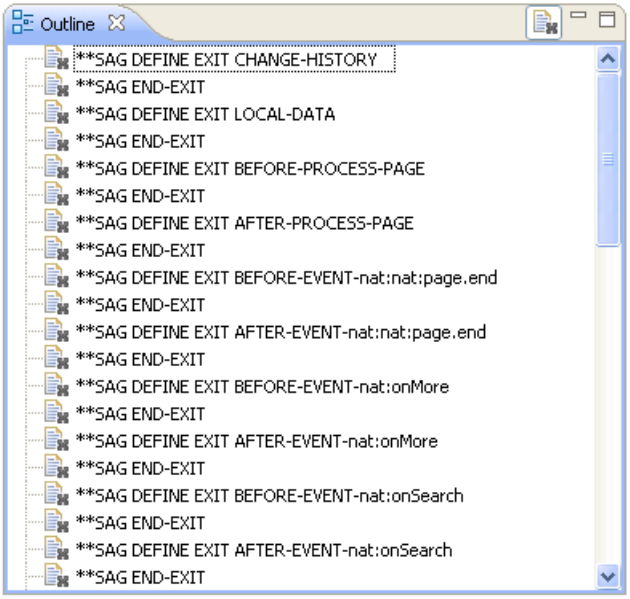
2 Select **Finish**.

The page is generated using the selected fields, column headings and search keys. The generated main program (.NSP extension) file is displayed in the editor. For example:

customer_page.xml | CUSTADAP.NS8 | CUSTMAIN.NSP

```
* >Natural Source Header 000000
**SAG GENERATOR: AJAX-BROWSE-MAIN-PROGRAM                    VERSION: 8.2.3
**SAG AdapterName: CUSTADAP
**SAG MainProgramName: CUSTMAIN
**SAG ObjectName: ACUSTN
**SAG PageName: customer_page
**SAG UIComponent: Component_A
**SAG KEY-PDA-NAME: ACUSTK
**SAG RESTRICTED-PDA-NAME: ACUSTP
**SAG ROW-PDA-NAME: ACUSTD
**SAG COLUMN_OBJECT: ACUSTD.CUSTOMER-NUMBER,true,Customer Number,true
**SAG COLUMN_OBJECT: ACUSTD.BUSINESS-NAME,true,Business Name,true
**SAG COLUMN_OBJECT: ACUSTD.PHONE-NUMBER,true,Phone Number,false
**SAG COLUMN_OBJECT: ACUSTD.M-STREET,true,M Street,false
**SAG COLUMN_OBJECT: ACUSTD.M-CITY,true,M City,false
**SAG COLUMN_OBJECT: ACUSTD.M-PROVINCE,true,M Province,false
**SAG COLUMN_OBJECT: ACUSTD.M-POSTAL-CODE,true,M Postal Code,false
**SAG COLUMN_OBJECT: ACUSTD.S-STREET,true,S Street,false
**SAG COLUMN_OBJECT: ACUSTD.S-CITY,true,S City,false
**SAG COLUMN_OBJECT: ACUSTD.S-PROVINCE,true,S Province,false
**SAG COLUMN_OBJECT: ACUSTD.S-POSTAL-CODE,true,S Postal Code,false
**SAG COLUMN_OBJECT: ACUSTD.CONTACT,true,Contact,false
**SAG COLUMN_OBJECT: ACUSTD.CREDIT-RATING,true,Credit Rating,false
**SAG COLUMN_OBJECT: ACUSTD.CREDIT-LIMIT,true,Credit Limit,false
**SAG COLUMN_OBJECT: ACUSTD.DISCOUNT-PERCENTAGE,true,Discount Percentage,false
**SAG COLUMN_OBJECT: ACUSTD.CUSTOMER-WAREHOUSE-ID,true,Customer Warehouse Id,true
**SAG COLUMN_OBJECT: ACUSTD.CUSTOMER-TIMESTAMP,true,Customer Timestamp,false
```

The available user exits are displayed in the **Outline** view. For example:



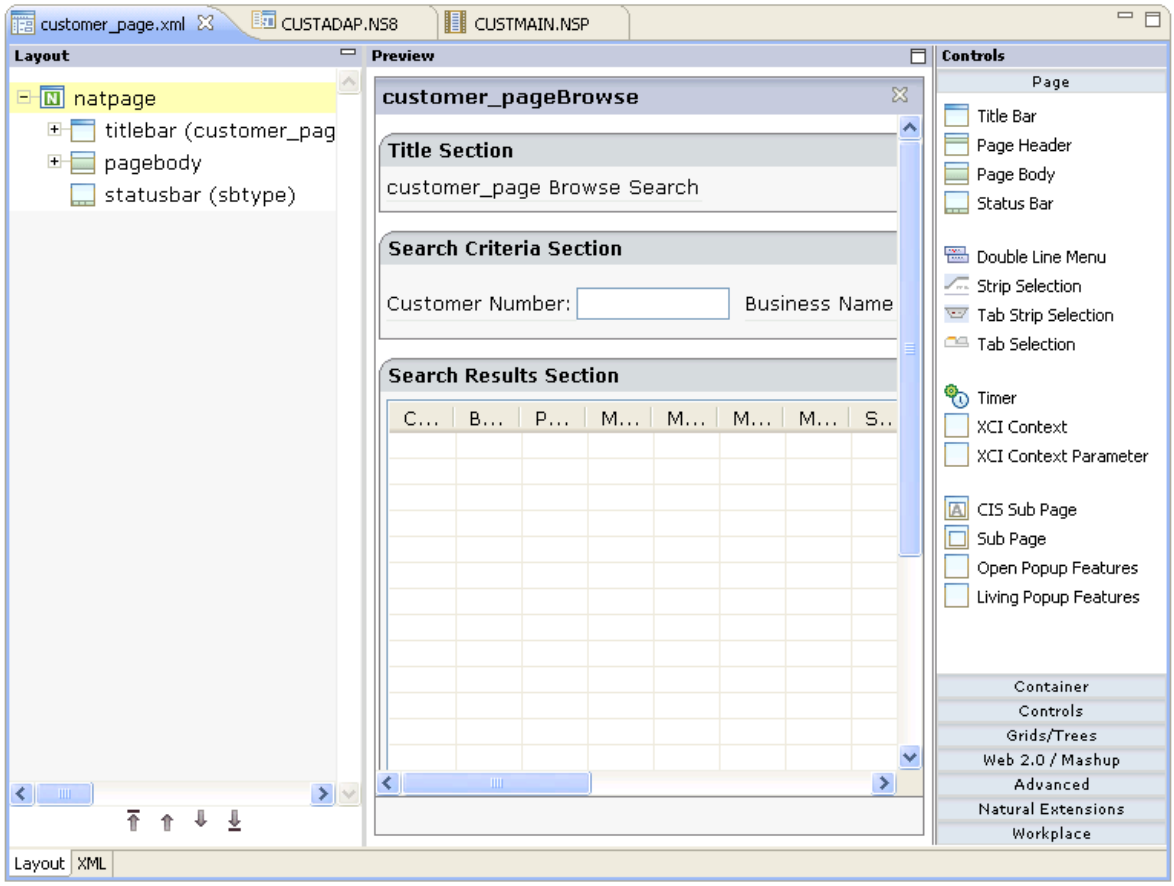
For every event in the page layout file, a BEFORE-EVENT and AFTER-EVENT user exit is generated (for example, "BEFORE-EVENT-nat:nat:page.end" above). When you add an event to the page layout file, a BEFORE-EVENT and AFTER-EVENT user exit is also generated. Code within these exits is preserved during regeneration.

The generated adapter (.NS8 extension) and page layout (.xml extension) files are also displayed in the editor. For example:

customer_page.xml | CUSTADAP.NS8 | CUSTMAIN.NSP

```
* >Natural Source Header 000000
* PAGE1: PROTOTYPE        --- CREATED BY Application Designer --- /*<RO>>
* PROCESS PAGE USING 'XXXXXXXX' WITH
* KEY.BUSINESS-NAME KEY.CUSTOMER-NUMBER KEY.CUSTOMER-WAREHOUSE-ID
* MOREVISIBLE SBLONGMESSAGE SBSHORTMESSAGE SBTYPE SEARCHVISIBLE
* UI.BUSINESS-NAME(*) UI.CONTACT(*) UI.COUNT(*) UI.CREDIT-LIMIT(*)
* UI.CREDIT-RATING(*) UI.CUSTOMER-NUMBER(*) UI.CUSTOMER-TIMESTAMP(*)
* UI.CUSTOMER-WAREHOUSE-ID(*) UI.DISCOUNT-PERCENTAGE(*) UI.M-CITY(*)
* UI.M-POSTAL-CODE(*) UI.M-PROVINCE(*) UI.M-STREET(*) UI.PHONE-NUMBER(*)
* UI.S-CITY(*) UI.S-POSTAL-CODE(*) UI.S-PROVINCE(*) UI.S-STREET(*)
* UI.UNIQUE-ID(*) UI.ROWCOUNT UI.SIZE UI.ASCENDING(*) UI.PROPNAME(*)
* UI.TOPINDEX
DEFINE DATA PARAMETER
/*( PARAMETER
1 KEY
2 BUSINESS-NAME (A30)
2 CUSTOMER-NUMBER (N5)
2 CUSTOMER-WAREHOUSE-ID (A3)
1 MOREVISIBLE (L)
1 SBLONGMESSAGE (A) DYNAMIC
1 SBSHORTMESSAGE (A) DYNAMIC
1 SBTYPE (A) DYNAMIC
1 SEARCHVISIBLE (L)
1 UI
2 UI-CUSTOMER_PAGE (1:*)
3 BUSINESS-NAME (A) DYNAMIC
3 CONTACT (A) DYNAMIC
3 COUNT (A) DYNAMIC
3 CREDIT-LIMIT (A) DYNAMIC
3 CREDIT-RATING (A) DYNAMIC
3 CUSTOMER-NUMBER (A) DYNAMIC
```

The following example shows the generated page layout file displayed in the **Layout** tab.



Note: For more information about these files, refer to the Natural for Ajax documentation.

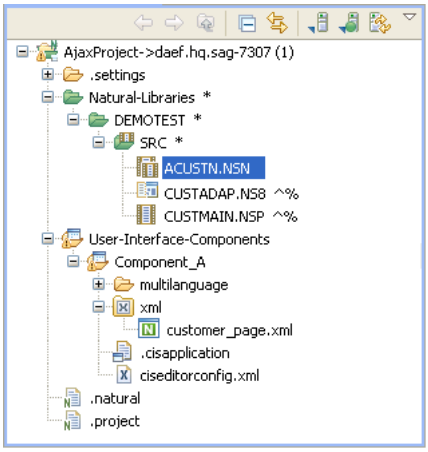
Select the **XML** tab to display the generated xml file. For example:

customer_page.xml | CUSTADAP.NS8 | CUSTMAIN.NSP

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <natpage xmlns:njx="http://www.softwareag.com/njx/njxMapConverter" natsinglebyte="true" natsource
3   <titlebar name="customer_pageBrowse" />
4   <pagebody>
5     <vdist height="10" />
6     <rowarea name="Title Section">
7       <tr>
8         <label name="customer_page Browse Search" />
9       </tr>
10      <vdist height="5" />
11    </rowarea>
12    <vdist height="10" />
13    <rowarea name="Search Criteria Section">
14      <vdist height="10" />
15      <itr>
16        <label name="Customer Number:" />
17        <field datatype="N 5" maxlength="20" valueprop="KEY.CUSTOMER-NUMBER" width="100" />
18        <hdist width="10" />
19        <label name="Business Name:" />
20        <field datatype="string 30" maxlength="20" valueprop="KEY.BUSINESS-NAME" width="100" />
21        <hdist width="10" />
22        <label name="Customer Warehouse Id:" />
23        <field datatype="string 3" maxlength="20" valueprop="KEY.CUSTOMER-WAREHOUSE-ID" width="10
24        <hdist width="10" />
25        <hdist width="10" />
26        <button method="onSearch" name="Search" visibleprop="SearchVisible" />
27        <hdist width="10" />
28      </itr>
29      <vdist height="10" />
30    </rowarea>
```

Layout | XML

The generated files are displayed in the **Project Explorer**. For example:



3 Open the context menu in the **Project Explorer** for the generated main program and adapter files.

4 Select **NaturalONE > Update**.

At this point, you can:

- Test the main program. For information, see *Test the Generated Main Program*.

- Define user exits. For information, see *Defining User Exits*.

Generate an Ajax Page for an Object-Maint Subprogram

This section describes how to generate an Ajax page for an object-maintenance subprogram.

The following topics are covered:

- Access the Ajax Object-Maint Page Wizard
- Specify Source and Target Details for the Ajax Object-Maint Page Wizard
- Configure Field Details for the Ajax Object-Maint Page Wizard

Access the Ajax Object-Maint Page Wizard

The section describes how to access the Ajax Object-Maint Page wizard. Before accessing the wizard, the following conditions must be met:

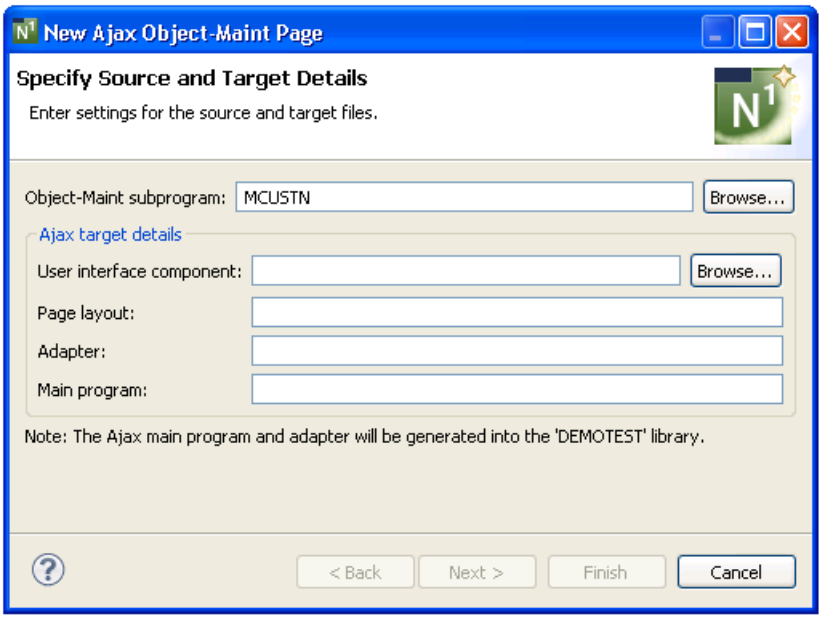
- The wizard must be started from a project that has been enabled for Ajax Developer.
- A user interface component must be available locally.

Note: For more information about the generated files and general Ajax architecture, refer to the Natural for Ajax documentation.

To access the Ajax Object-Maint Page wizard

1. Open the context menu for a subprogram that was generated by either the Object-Maint-N1 or Object-Maint-Subp generator.
2. Select **Code Generation > New Ajax Object-Maint Page**.

   The **Specify Source and Target Details** panel is displayed. For example:



Specify Source and Target Details for the Ajax Object-Maint Page Wizard

To specify source and target details

1 Define the following parameters:

Parameter	Description	Required/Optional/Conditional
Object-Maint subprogram	Name of the Object-Maint-Subp-generated subprogram for which you are creating the page. To change the name of the subprogram, either type the name of a new subprogram in **Object-Maint subprogram** or select **Browse** to display the available subprograms for selection.	Required
User interface component	Name of the user interface component for the page. Either type the name of an existing component in **User interface component** or select **Browse** to display the available components for selection.	Required
Page layout	Name of the page layout file to be generated. This name must follow standard xml naming conventions (do not include the .xml extension). **Tip:** Avoid using spaces in the page name as it may cause problems during generation.	Required
Adapter	Name of the adapter file to be generated. This name must follow standard Natural naming conventions (do not include the .NS8 extension).	Required

Parameter	Description	Required/Optional/Conditional
Main program	Name of the main program file to be generated. This name must follow standard Natural naming conventions (do not include the .NSP extension).	Required

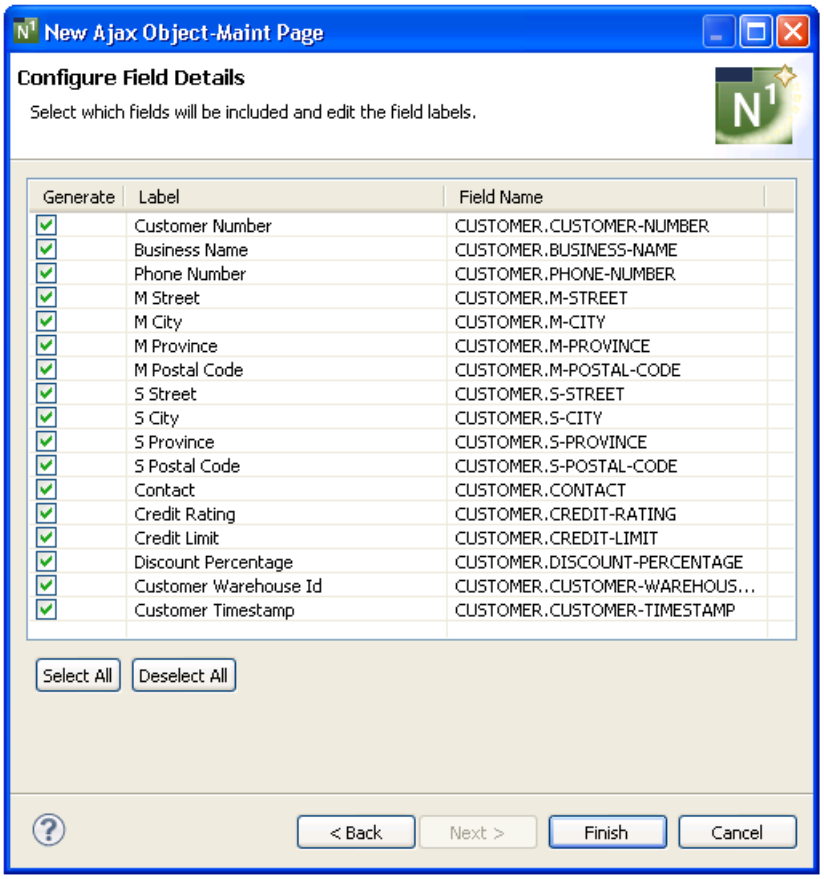
2 Select **Finish**.

The page is generated using all fields in the PDA, as well as all default field labels.

Or:

Select **Next**.

The **Configure Field Details** panel is displayed. For example:



Note: Array fields are not currently supported.

Configure Field Details for the Ajax Object-Maint Page Wizard

This panel allows you to select which fields are generated for the page and what labels will be displayed. The **Field Name** column displays the fully qualified name of each field in the PDA; these parameters are read-only.

To configure field details

1 Define the following parameters:

Parameter	Description	Required/Optional/Conditional
Generate	Indicates whether the corresponding field is generated or not. To exclude a field, deselect **Generate** for that field. A minimum of one field must be selected.	Optional
Label	Label displayed on the generated page for the corresponding field. You can change this label as desired.	Optional

2 Select **Finish**.

The page is generated using the selected fields, column headings and search keys. The generated main program (.NSP extension) file is displayed in the editor. For example:

```
cust_maint.xml   CUSTADPT.NS8   CUSTMMP.NSP
* >Natural Source Header 000000
**SAG GENERATOR: AJAX-MAINT-MAIN-PROGRAM                    VERSION: 8.2.3
**SAG AdapterName: CUSTADPT
**SAG MainProgramName: CUSTMMP
**SAG ObjectName: MCUSTN
**SAG PageName: cust_maint
**SAG UIComponent: Component_A
**SAG GUI-FIELD: CUSTOMER.CUSTOMER-NUMBER,true,Customer Number
**SAG GUI-FIELD: CUSTOMER.BUSINESS-NAME,true,Business Name
**SAG GUI-FIELD: CUSTOMER.PHONE-NUMBER,true,Phone Number
**SAG GUI-FIELD: CUSTOMER.M-STREET,true,M Street
**SAG GUI-FIELD: CUSTOMER.M-CITY,true,M City
**SAG GUI-FIELD: CUSTOMER.M-PROVINCE,true,M Province
**SAG GUI-FIELD: CUSTOMER.M-POSTAL-CODE,true,M Postal Code
**SAG GUI-FIELD: CUSTOMER.S-STREET,true,S Street
**SAG GUI-FIELD: CUSTOMER.S-CITY,true,S City
**SAG GUI-FIELD: CUSTOMER.S-PROVINCE,true,S Province
**SAG GUI-FIELD: CUSTOMER.S-POSTAL-CODE,true,S Postal Code
**SAG GUI-FIELD: CUSTOMER.CONTACT,true,Contact
**SAG GUI-FIELD: CUSTOMER.CREDIT-RATING,true,Credit Rating
**SAG GUI-FIELD: CUSTOMER.CREDIT-LIMIT,true,Credit Limit
**SAG GUI-FIELD: CUSTOMER.DISCOUNT-PERCENTAGE,true,Discount Percentage
**SAG GUI-FIELD: CUSTOMER.CUSTOMER-WAREHOUSE-ID,true,Customer Warehouse Id
**SAG GUI-FIELD: CUSTOMER.CUSTOMER-TIMESTAMP,true,Customer Timestamp
************************************************************************
* Program  : CUSTMMP
* System   : DEMOTEST
* Title    : AJAX main program for: MCUSTN
*
* Generated: Mon Aug 29 17:22:47 EDT 2011
* Function : This program invokes the Object MAINT subprogram
```

The available user exits are displayed in the **Outline** view. For example:

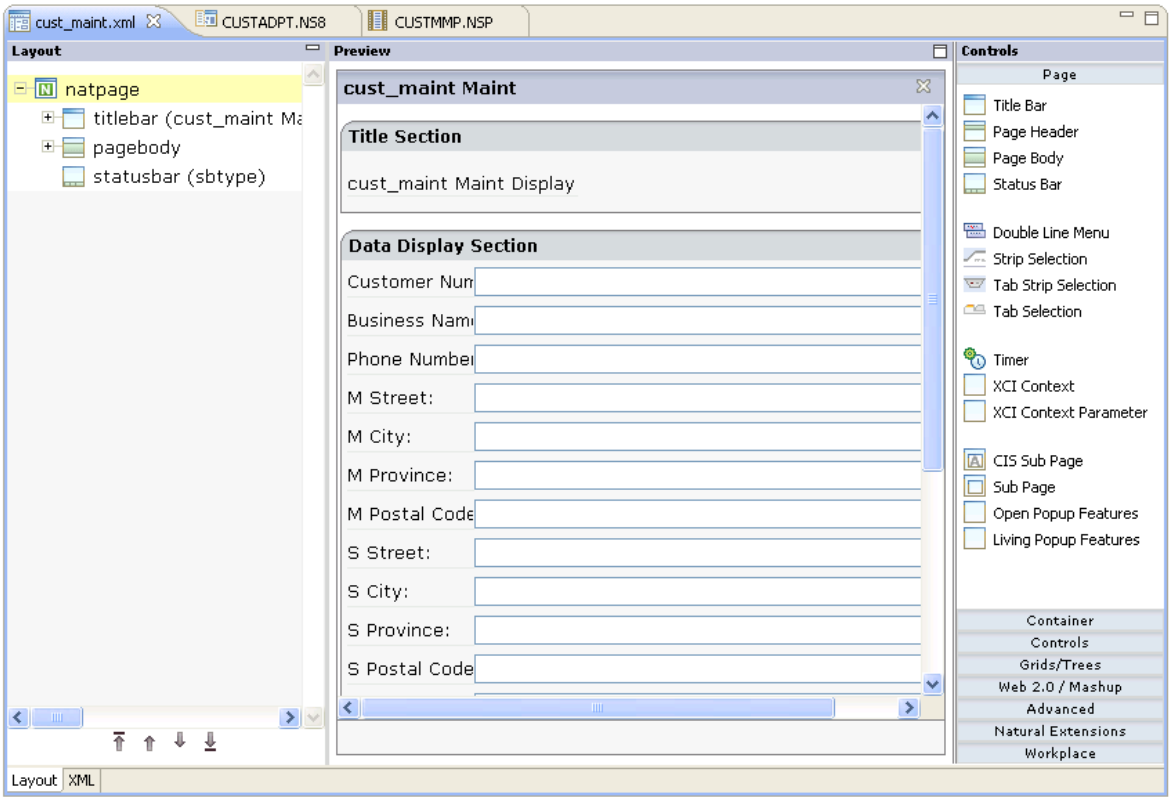
```
Outline
**SAG DEFINE EXIT CHANGE-HISTORY
**SAG END-EXIT
**SAG DEFINE EXIT LOCAL-DATA
**SAG END-EXIT
**SAG DEFINE EXIT BEFORE-PROCESS-PAGE
**SAG END-EXIT
**SAG DEFINE EXIT AFTER-PROCESS-PAGE
**SAG END-EXIT
**SAG DEFINE EXIT BEFORE-EVENT-nat:nat:page.end
**SAG END-EXIT
**SAG DEFINE EXIT AFTER-EVENT-nat:nat:page.end
**SAG END-EXIT
**SAG DEFINE EXIT BEFORE-EVENT-nat:onAdd
**SAG END-EXIT
**SAG DEFINE EXIT AFTER-EVENT-nat:onAdd
**SAG END-EXIT
**SAG DEFINE EXIT BEFORE-EVENT-nat:onDelete
**SAG END-EXIT
**SAG DEFINE EXIT AFTER-EVENT-nat:onDelete
**SAG END-EXIT
```

For every event in the page layout file, a BEFORE-EVENT and AFTER-EVENT user exit is generated (for example, "BEFORE-EVENT-nat:nat:page.end" above). When you add an event to the page layout file, a BEFORE-EVENT and AFTER-EVENT user exit is also generated. Code within these exits is preserved during regeneration.

The generated adapter (.NS8 extension) and page layout (.xml extension) files are also displayed in the editor. For example:

```
cust_maint.xml   CUSTADPT.NS8   CUSTMMP.NSP
* >Natural Source Header 000000
* PAGE1: PROTOTYPE        --- CREATED BY Application Designer --- /*<RO>>
* PROCESS PAGE USING 'XXXXXXXX' WITH
* ADDVISIBLE DELETEVISIBLE GETVISIBLE NEXTVISIBLE PREVIOUSVISIBLE
* SAVEVISIBLE SBLONGMESSAGE SBSHORTMESSAGE SBTYPE UI.BUSINESS-NAME
* UI.CONTACT UI.CREDIT-LIMIT UI.CREDIT-RATING UI.CUSTOMER-NUMBER
* UI.CUSTOMER-TIMESTAMP UI.CUSTOMER-WAREHOUSE-ID UI.DISCOUNT-PERCENTAGE
* UI.M-CITY UI.M-POSTAL-CODE UI.M-PROVINCE UI.M-STREET UI.PHONE-NUMBER
* UI.S-CITY UI.S-POSTAL-CODE UI.S-PROVINCE UI.S-STREET
DEFINE DATA PARAMETER
/*( PARAMETER
1 ADDVISIBLE (L)
1 DELETEVISIBLE (L)
1 GETVISIBLE (L)
1 NEXTVISIBLE (L)
1 PREVIOUSVISIBLE (L)
1 SAVEVISIBLE (L)
1 SBLONGMESSAGE (A) DYNAMIC
1 SBSHORTMESSAGE (A) DYNAMIC
1 SBTYPE (A) DYNAMIC
1 UI
2 XMLCUSTOMER
3 BUSINESS-NAME (A30)
3 CONTACT (A30)
3 CREDIT-LIMIT (P11.2)
3 CREDIT-RATING (A3)
3 CUSTOMER-NUMBER (N5)
3 CUSTOMER-TIMESTAMP (T)
3 CUSTOMER-WAREHOUSE-ID (A3)
3 DISCOUNT-PERCENTAGE (P3.2)
3 M-CITY (A20)
```

The following example shows the generated page layout file in the **Layout** tab.



Note: For more information about these files, refer to the Natural for Ajax documentation.

Select the **XML** tab to display the generated xml file. For example:

```
cust_maint.xml   CUSTADPT.NS8   CUSTMMP.NSP
1  <?xml version="1.0" encoding="UTF-8"?>
2  <natpage xmlns:njx="http://www.softwareag.com/njx/njxMapConverter" natsinglebyte="true" natsourc
3    <titlebar name="cust_maint Maint" />
4    <pagebody>
5      <vdist height="10" />
6      <rowarea name="Title Section">
7        <vdist height="10" />
8        <itr>
9          <label name="cust_maint Maint Display" />
10         <hdist width="10" />
11       </itr>
12       <vdist height="10" />
13     </rowarea>
14     <vdist height="10" />
15     <rowarea name="Data Display Section">
16       <tr>
17         <label name="Customer Number:" nowrap="true" width="5%" />
18         <field datatype="N 5" maxlength="5" name="UI.XMLCUSTOMER.CUSTOMER-NUMBER" njx:natname="U
19       </tr>
20       <vdist height="5" />
21       <tr>
22         <label name="Business Name:" nowrap="true" width="5%" />
23         <field datatype="string 30" maxlength="30" name="UI.XMLCUSTOMER.BUSINESS-NAME" njx:natna
24       </tr>
25       <vdist height="5" />
26       <tr>
27         <label name="Phone Number:" nowrap="true" width="5%" />
28         <field datatype="N 10" maxlength="10" name="UI.XMLCUSTOMER.PHONE-NUMBER" njx:natname="UI
29       </tr>
30       <vdist height="5" />
Layout  XML
```

The generated files are displayed in the **Project Explorer**. For example:

```
AjaxProject->daef.hq.sag-7307 (1)
  .settings
  Natural-Libraries *
    DEMOTEST *
      SRC *
        MORDN.NSN
        CUSTADPT.NS8 ^%
        CUSTMMP.NSP ^%
  User-Interface-Components
    Component_A
      multilanguage
      xml
        cust_maint.xml
      .cisapplication
      ciseditorconfig.xml
  .natural
  .project
```

3 Open the context menu in the **Project Explorer** for the generated main program and adapter files.

4 Select **NaturalONE > Update**.

At this point, you can:

- Test the main program. For information, see *Test the Generated Main Program*.

- Define user exits. For information, see *Defining User Exits*.

Generate an Ajax Main Program from an Adapter File

There are two ways to create an Ajax main program:

- Using standard Natural for Ajax functionality.
- Using the Ajax Main Program wizard and an existing adapter file (.NS8 extension). This wizard creates a main program that is similar to the standard one, except it includes support for user exits (for protected code) and regeneration when the Ajax UI changes (which allows the fields on a page to be updated without overwriting the user interface logic).

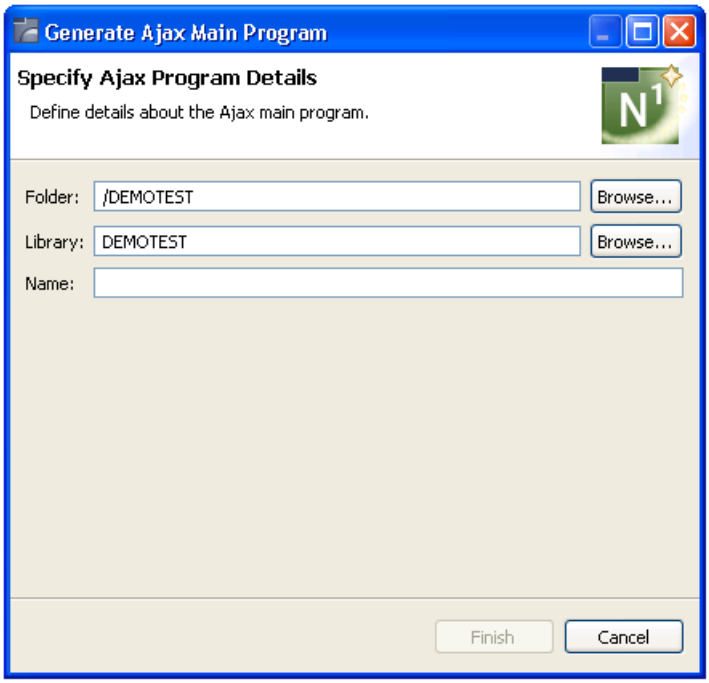
This section describes how to use the Ajax Main Program wizard to generate a main program from an adapter file, as well as how to regenerate the generated program.

To generate an Ajax main program from an adapter file

1 Open the context menu for the adapter file in the **Project Explorer**.

2 Select **Code Generation > Generate Ajax Main Program**.

The **Specify Ajax Program Details** panel is displayed. For example:



3 Type the name of the main program in **Name**.

Optionally, you can:

Task	Procedure
Select a folder in which to generate the program.	Type the name of the folder in **Folder** or select **Browse** to display a window listing the existing folders for selection. The folder must currently exist within the selected Ajax project. **Note:** This option allows you to generate modules into more complex library structures (for example, "Natural-Libraries/*my library* (MYLIB)/SRC"). When this option is not specified, the modules will be generated into the basic library folder (for example, "Natural-Libraries/MYLIB/SRC", "Natural-Libraries/MYLIB/Subprograms", etc.).
Select a library in which to generate the program.	Type the name of the library in **Library** or select **Browse** to display a window listing the existing libraries for selection. The library must currently exist. **Note:** The libraries listed for selection are based on the current project.

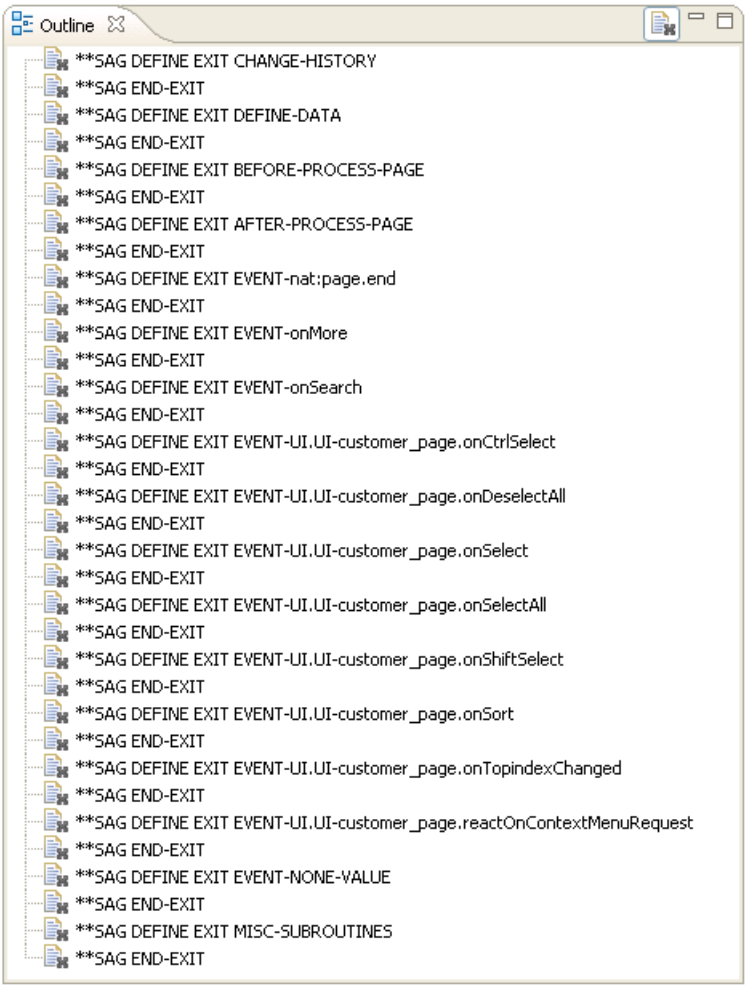
4 Select **Finish** to generate the main program.

The generated main program is displayed in the editor view. For example:

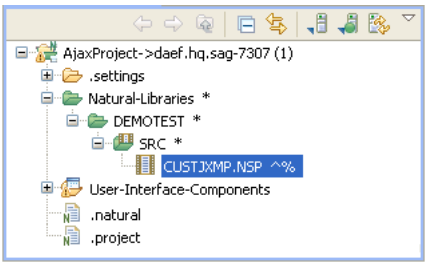
```
CUSTJXMP.NSP
* >Natural Source Header 000000
**SAG GENERATOR: NJXProgramGenerator                    VERSION: 8.2.4
**SAG adapter: CUSTADAP.NS8
**SAG DESCS(1): NJX Program to execute CUSTADAP.NS8
************************************************************************
* Program  : CUSTJXMP
* System   : DEMOTEST
* Title    : Module ...
* Generated: Thu Feb 02 16:34:51 EST 2012
* Function : NJX Program to execute CUSTADAP.NS8
*
*
*
* History
**SAG DEFINE EXIT CHANGE-HISTORY
**SAG END-EXIT
************************************************************************

DEFINE DATA LOCAL
/* PARAMETERS from parsed adapter
1 KEY
  2 BUSINESS-NAME (A30)
  2 CUSTOMER-NUMBER (N5)
  2 CUSTOMER-WAREHOUSE-ID (A3)
1 MOREVISIBLE (L)
1 SBLONGMESSAGE (A) DYNAMIC
1 SBSHORTMESSAGE (A) DYNAMIC
1 SBTYPE (A) DYNAMIC
1 SEARCHVISIBLE (L)
1 UI
  2 UI-CUSTOMER_PAGE (1:*)
    3 BUSINESS-NAME (A) DYNAMIC
    3 CUSTOMER-NUMBER (A) DYNAMIC
    3 PHONE-NUMBER (A) DYNAMIC
  2 UI-CUSTOMER_PAGEINFO
    3 ROWCOUNT (I4)
    3 SIZE (I4)
    3 SORTPROPS (1:*)
      4 ASCENDING (L)
```

The available user exits are displayed in the **Outline** view. For example:



The generated program is also displayed in the **Project Explorer**. For example:



5 Open the context menu for the generated main program in the **Project Explorer**.

6 Select **NaturalONE > Update**.

At this point, you can:

- Test the main program. For information, see *Test the Generated Main Program*.

- Define user exits. For information, see *Defining User Exits*.

Regenerate the Ajax Main Program

This section describes how to regenerate the main program file (.NSP extension) that was generated using the Ajax Main Program wizard.

To regenerate the main program

1. Open the context menu for the program in the **Project Explorer**.
2. Select **Code Generation > Regenerate Using Wizard**.

   The **Specify Ajax Program Details** panel is displayed. After selecting **Finish**, the main program is regenerated.

   Or:

   Select **Code Generation > Regenerate**.

   The main program is regenerated without displaying the wizard panel.

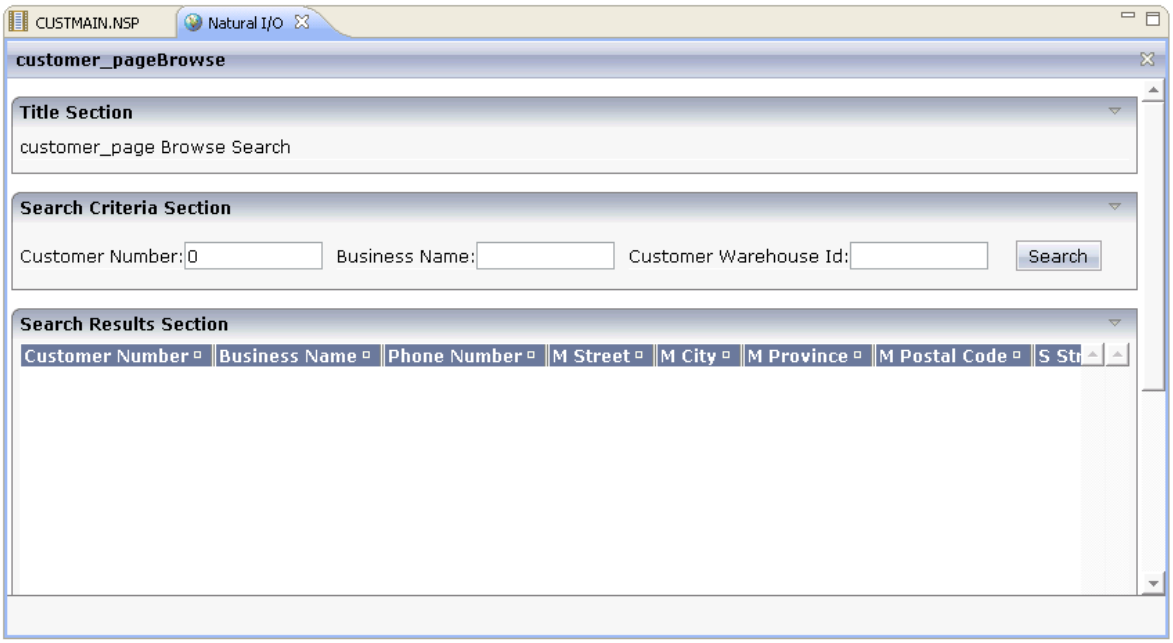
   Note: You can use standard selection techniques to select more than one file.

Test the Generated Main Program

To test the generated main program

1. Open the context menu in the **Project Explorer** view for the generated main program file (.NSP extension).
2. Select **NaturalONE > Execute**.

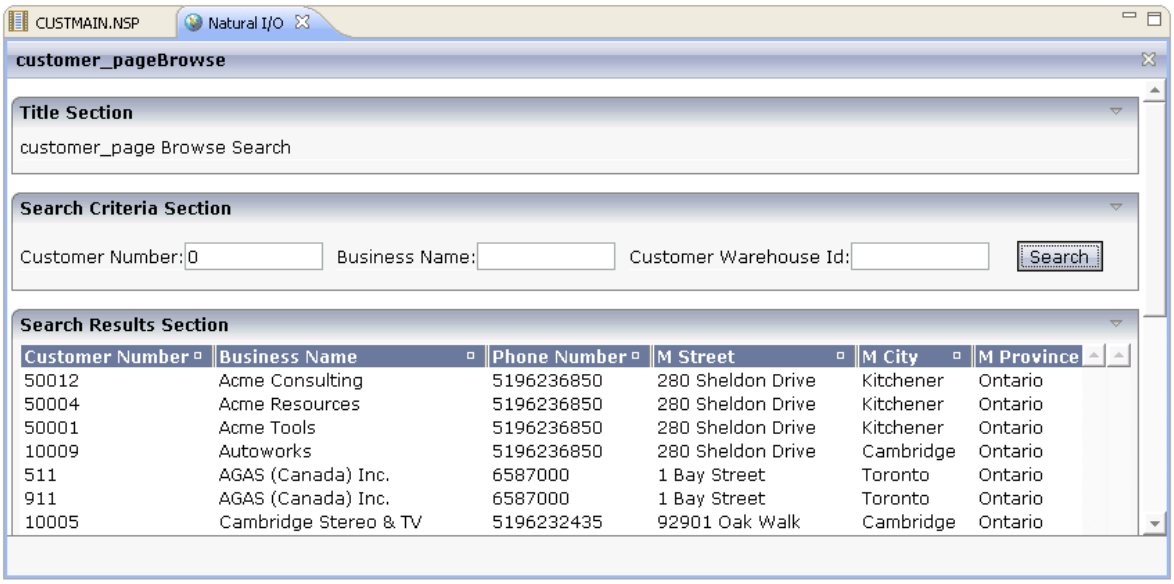
   The page is displayed in the editor. For example:



By default, the generated page displays row data in a table, where each field in the object (row) PDA is a column. All search key fields are displayed at the top of the page.

3 Select **Search**.

The results of the search are displayed. For example:



Regenerate the Main Program

This section describes how to regenerate the main program file (.NSP extension).

To regenerate the main program

1 Open the context menu for the main program in the **Project Explorer**.

2 Select **Code Generation > Regenerate Using Wizard**.

For information, see *Regenerate Using Wizard*.

Or:

Select **Code Generation > Regenerate**.

For information, see *Regenerate*.

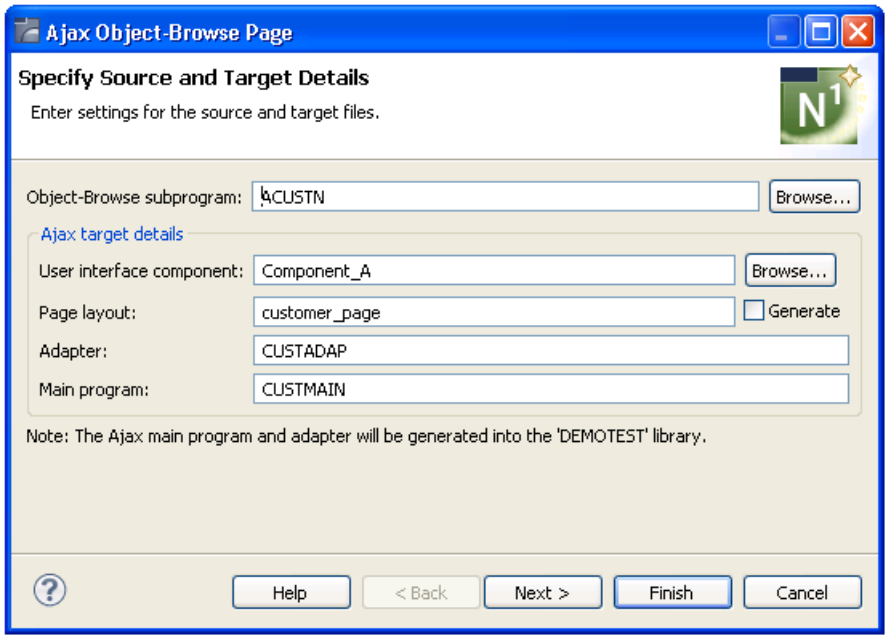
Regenerate Using Wizard

Use this option when you want to make changes to the wizard parameters before regenerating the Ajax page.

To regenerate using the wizard panel(s)

1 Select **Code Generation > Regenerate Using Wizard**.

The first specification panel for the wizard is displayed. For example:



2 Edit the specifications as desired.

If the page layout file (.xml extension) has changed since the previous generation, you can select **Generate** to regenerate the page layout file as well.

3 Select **Finish** to regenerate the main program, adapter and, optionally, the page layout files.

4 Open the context menu in the **Project Explorer** for the regenerated main program and adapter files.

5 Select **NaturalONE > Update**.

Regenerate

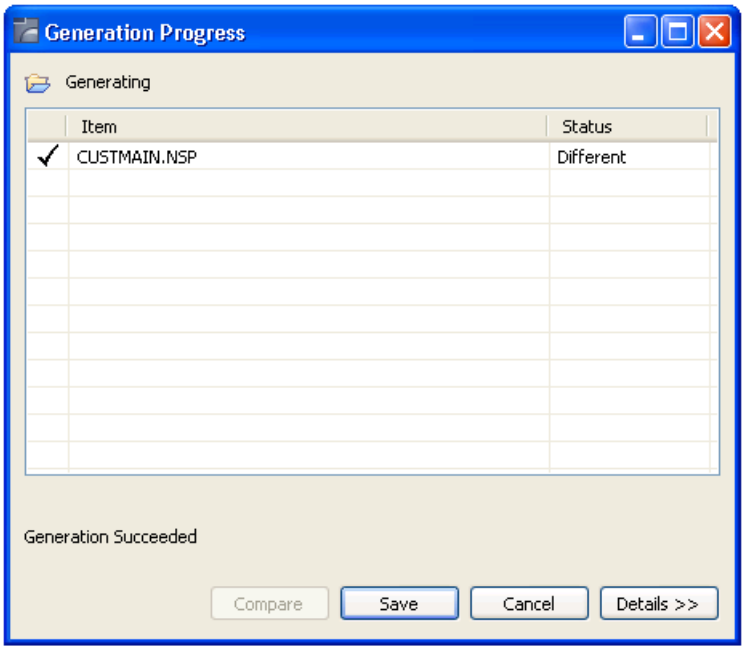
Use this option when the object-browse or object-maintenance subprogram has changed and you want to incorporate the changes in the Ajax page.

To regenerate without using the wizard panels

1 Select **Code Generation > Regenerate**.

Note: You can use standard selection techniques to select more than one file.

The main program (.NSP) file is regenerated without displaying the wizard panels and the **Generation Progress** window is displayed. For example:



2 Select **Save** to save the details.

3 Open the context menu in the **Project Explorer** for the regenerated file(s).

4

5 Select **NaturalONE > Update**.