General Installation Details

Version 2024.2
2024-09-05

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

General Installation Details
PDF generated on 2024-09-05
InterSystems IRIS® Version 2024.2

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Tables

1

Installation Directory

The installation directory is the directory in which an InterSystems IRIS instance is installed. In general, this directory should be empty before installation. Throughout the documentation, this is referred to as *install-dir*. Once InterSystems IRIS is installed, it is impossible to change the installation directory.

There are several restrictions to what you can specify as the *install-dir*. The directory must be a fully resolved physical path, containing no symbolic links. The name of the directory can only use characters in the US ASCII character set, and cannot contain a caret (^). Also, InterSystems IRIS *cannot* be installed into a directory that is:

- a UNC (non-local) path.
- at the root level of a drive (such as C:\).
- anywhere under the \Program Files directory.

If unspecified during installation, *install-dir* uses a default value. This default varies by platform, installation type, and user choice, as shown in the following table:

Platform	Installation Type	Default Directory
Windows	attended	C:\InterSystems\Iris (or Iris*N* when multiple instances exist), unless installing user specifies otherwise.
	unattended	C:\InterSystems\Iris (or Iris*N* when multiple instances exist), unless `INSTALLDIR` property specifies otherwise.
UNIX®, Linux, macOS	attended	No default; installing user must specify. Do not choose the /home directory, any of its subdirectories, or the /usr/local/etc/irissys directory.
	unattended	No default; `ISC_PACKAGE_INSTALLDIR` parameter required.

1.1 Reusing Installation Files

In general, you should install InterSystems IRIS to an empty directory. If you specify a non-empty directory (for example, the directory of a different, previously uninstalled instance), your new installation of InterSystems IRIS automatically reuses any relevant files, like iris.cpf and your license file. While this can be useful for certain use cases like configuring failover clusters, this can cause issues if these files are invalid or incompatible with your new installation. For example,

installation will fail if the license is expired, and your instance will not start if there are invalid configuration parameters in iris.cpf.

2

Setup Types

During installation, you can choose which components of InterSystems IRIS you would like to install. The options are:

2.1 Setup Types

- **Development** — Includes the InterSystems IRIS Database Engine (User Database, Language Gateways, Server Monitoring Tools), Web Gateway, all supported language bindings, and database drivers. Select this option if you plan to use this instance to perform both client and server tasks.
- **Server** — Includes InterSystems IRIS Database Engine (User database, Language Gateways, Server monitoring tools) and Web Gateway. Select this option if you plan to use this instance as an InterSystems IRIS database server which can be accessed by InterSystems IRIS clients.
- **Custom** — Allows user to specify specific components to install or uninstall. Select this option if you plan to install or remove specific InterSystems IRIS components.

On Windows, you can choose from these two additional setup types:

- **Client** — Includes ODBC and JDBC drivers and the InterSystems IRIS Application Development components. Select this option if you plan to use this instance as a client to an InterSystems IRIS database server on this or another computer.
- **Web Server** — Includes Web Gateway. Select this option if you want to install only those parts of InterSystems IRIS that are required for the Web Gateway.

The following table identifies which components are installed for each setup type. When performing a custom installation, you may include only individual components from a group.

Table 2–1: Components Installed by Setup Type

Component Group	Components	Development	Server	Client	Web
InterSystems IRIS Database Engine (InterSystems IRIS Server)	Server Monitoring Tools User database Language Gateways Agent Service (ISCAgent) Apache FOP (Formatting Objects Processor) InterSystems IntegratedML	X	X		
InterSystems IRIS launcher (on Windows only)		X	X	X	
Database Drivers	ODBC Driver Java Database Connectivity	X		X	
InterSystems IRIS Application Development	Java Binding for InterSystems IRIS InterSystems IRIS shared library support .NET Binding for InterSystems IRIS Threaded Server Libraries Other Samples	X		X	
Web Gateway	Web Gateway binaries for IIS Web Gateway binaries for Apache 2.4.x	X	X		X

2.2 Preexisting CSP Gateway

If the CSP Gateway is already installed on your system when you install the Web Gateway, the installer automatically upgrades the CSP Gateway to the Web Gateway. However, the installer creates a new copy of the CSP.ini file. To preserve the CSP.ini file from the CSP Gateway, perform the following steps:

1. Create a backup of the CSP.ini file, located in *install-dir*/CSP/bin.
2. Execute the Web Gateway installer.
3. Restore the backup of CSP.ini.

Note: Installing the Web Gateway does not close any old connections. To remove these old connections, you must manually close them from the System Status page.

3

Character Width Setting

You must choose between 8-bit or Unicode (16-bit) character width for your installation. An 8-bit instance cannot process data in 16-bit format, while a Unicode instance can process both 8-bit and 16-bit data.

If you need your instance to store and process data in a language that uses only Unicode character sets, such as Japanese, you must select Unicode. If the base character set for the locale your instance uses is based on the Latin-1 character set, ISO8859-1, you can select 8-bit, but bear in mind:

- If you select 8-bit, the data stored by your instance is portable only to 8-bit locales based on the same character set.
- If you select Unicode, the data stored by the instance is not portable to an 8-bit instance.

4

Port Numbers

A standard installation sets the following port numbers for an InterSystems IRIS instance:

- *Superserver port number* — `1972`. If taken, then `51773` or the first available subsequent number.
- *Telnet* — 23

You can assign different port numbers during a new installation by performing a **Custom** installation (see Setup Type). You cannot enter a port number greater than `65535`.

For information about setting the port numbers on an already-installed instance of InterSystems IRIS, see Set Port Numbers.

5

Creating and Using an Installation Manifest

This topic describes how to use the %Installer utility to create an installation manifest that describes a specific InterSystems IRIS® data platform configuration and use it to generate code to configure an InterSystems IRIS instance.

5.1 Overview of an Installation Manifest

The %Installer utility lets you define an installation manifest that describes and configures a specific InterSystems IRIS configuration, rather than a step-by-step installation process. To do so, you create a class that contains an XData block describing the configuration you want, using variables that contain the information usually provided during installation (superserver port, operating system, and so on). You also include in the class a method that uses the XData block to generate code to configure the instance. This appendix provides installation manifest examples that you can copy and paste to get started.

Once you have defined the manifest, you can call it during installation, from a Terminal session, or from code. The manifest must be run in the %SYS namespace.

5.2 Creating an Installation Manifest

This section contains the information needed to create an installation manifest, broken into the following subjects:

- Manifest Class Definition
- Common Manifest Options
- Variables Within <Manifest> Tags
- List of <Manifest> Tags

5.2.1 Manifest Class Definition

To create a class that defines an installation manifest, create a class that meets the following criteria (or start from the blank template in the Installation Manifest Examples section below):

- The class must include the %occInclude include file.
- The class must contain an XData block specifying the details of the installation. The name of the XData block will be used later as an argument. The root element in the XData block must be `<Manifest>`. For details, see "List of <Manifest> Tags".
- The class must define a **setup()** method that refers to the XData block by name, as seen in the Installation Manifest Examples section.

5.2.2 Common Manifest Options

This section describes some of the common tasks an installation manifest is used to perform. The following options are described:

- Define Namespaces
- Add Users and Passwords
- Write to the Manifest Log
- Perform Post-Upgrade Tasks

5.2.2.1 Define Namespaces

To define namespaces, add any number of `<Namespace>` tags within the `<Manifest>` tag.

If you want to define databases or mappings for a namespace, put a `<Configuration>` tag inside the `<Namespace>` tag. For each database the namespace contains, add a `<Database>` tag within the `<Configuration>` tag. To define mappings, add `<GlobalMapping>`, `<RoutineMapping>`, and `<ClassMapping>` tags beneath the appropriate `<Database>` tag. The `</Configuration>` tag activates the defined mappings.

Also within a `<Namespace>` tag, you can load globals, routines, and classes using the `<Import>` tag. You can also invoke class methods, which in turn can execute routines and access globals that have been imported, with the `<Invoke>` tag.

You can define variables with the `<Var>` tag. To reference a variable within the manifest, use the `${var_name}` syntax. For more information, see Variables Within <Manifest> Tags below.

5.2.2.2 Add Users and Passwords

There are multiple ways to add users (including their roles and passwords) to the installed instance:

- Include a `<User>` tag in an installation manifest, as described in the List of <Manifest> Tags.

  The *PasswordVar* parameter of the `<User>` tag specifies the variable containing the password for the user; for example, by defining `PasswordVar="Pwd"`, you are specifying that the value of the variable *Pwd* is the password for a user. There are a variety of ways to populate this variable, but it is ultimately up to you to do this. You might consider using a remote method invocation to another instance of InterSystems IRIS or a web service; the problem with these approaches is that the server that is installing InterSystems IRIS may need internet access. Other possibilities include importing the method you are using to the instance you are installing, or adding a client-side form to the install that prompts for users and passwords, which can be passed to your manifest.
- Edit users in the Management Portal after installation is complete, as described in Users.
- Using the Security.Users class on a staging instance of InterSystems IRIS, as follows:

  1. Export the user information by using the **Security.Users.Export()** method.

2. Import the user information by adding the following at the beginning of your manifest class (in the %SYS namespace):

```
<Invoke Class="Security.Users" Method="Import" CheckStatus="true">
<Arg Value="PathToExportedUserInformation"/>
</Invoke>
```

where *PathToExportedUserInformation* is the location of the output file specified in the **Security.Users.Export()** method.

5.2.2.3 Write to the Manifest Log

You can define messages to be added to the manifest log by incorporating <Log> tags in your class, in the following format:

```
<Log Level="<level>" Text="<text>"/>
```

The log level must be in the range of -1 to 3, where -1 is "none" and 3 is "verbose"; if the log level specified in the **setup()** method is equal to or greater than the Level property in the <Log> tag, the message is logged. The text is limited to 32,000 characters.

You set the log level via the second argument of the **setup()** method (for more information, see "Using the Manifest" later in this appendix). Since this cannot be passed to the manifest from the install, you must set it in the class as follows:

```
ClassMethod setup(ByRef pVars, pLogLevel As %Integer = 3,
   pInstaller As %Installer.Installer,
   pLogger As %Installer.AbstractLogger)
   As %Status [ CodeMode = objectgenerator, Internal ]
```

You can direct where messages are displayed via the fourth argument (%Installer.AbstractLogger) of the **setup()** method; for example, if you instantiate a %Installer.FileLogger object referencing an operating system file, all %Installer and log messages are directed to that file. (Without this argument, all messages are written to the primary device if **setup()** is called directly, and are ignored if it is executed by the installer.)

5.2.2.4 Perform Post-Upgrade Tasks

When upgrading an instance of InterSystems IRIS, you can use an installation manifest to automatically perform any necessary post-upgrade tasks, such as recompiling code. You do this by using the <Invoke> tag to call the class methods described in the How to Compile Namespaces section of the "Upgrading InterSystems IRIS" chapter of this book.

See the Installation Manifest Examples section of this appendix for an example of a manifest that can be used during an upgrade.

5.2.3 Variables Within <Manifest> Tags

Some tags can contain expressions (strings) that are expanded when the manifest is executed. There are three types of expressions that can be expanded, as follows:

${<var_name>} — Variables

Expands to the value of the variable. In addition to the predefined variables described in this section, you can specify additional variables with the <Var> tag.

${#<param_name>} — Class Parameters

Expands to the value of the specified parameter for the manifest class.

#{<ObjectScript_expression>} — ObjectScript Expressions

Expands to the specified InterSystems IRIS Object Script expression (which must be properly quoted).

Note: Parameter expressions are expanded at compile time, which means they can be nested within variable and ObjectScript expressions. Additionally, variable expressions are expanded before ObjectScript expressions and thus can be nested within the latter.

The following table lists the predefined variables that are available to use in the manifest:

Variable Name	Description
SourceDir	(Available only when the installer is run) Directory from which the installation (setup_irisdb.exe or irisinstall) is running.
ISCUpgrade	(Available only when the installer is run) Indicates whether this is a new installation or an upgrade. This variable is either `0` (new installation) or `1` (upgrade).
CFGDIR	See *INSTALLDIR*.
CFGNAME	Instance name.
CPUCOUNT	Number of operating system CPUs.
CSPDIR	CSP directory.
HOSTNAME	Name of the host server.
INSTALLDIR	Directory into which InterSystems IRIS is installed.
MGRDIR	Manager (mgr) directory.
PLATFORM	Operating system.
PORT	InterSystems IRIS superserver port.
PROCESSOR	Processor chip.
VERSION	InterSystems IRIS version number.

5.2.4 List of <Manifest> Tags

All the information for code generation is contained in the outermost XML tag, `<Manifest>`. The tags within `<Manifest>` describe a specific configuration of InterSystems IRIS. This section contains a list of these tags and their function. Not all tags and attributes are listed here; for a complete list, see the %Installer class reference documentation. The default value for an attribute, if any, is listed in square brackets next to each attribute.

Important: Null arguments cannot be passed in tags in a manifest class. For example, the `<Arg/>` tag passes an empty string, equivalent to `<Arg=""/>`, not null.

- <Arg>
- <ClassMapping>
- <Compile>
- <Configuration>
- <CopyClass>
- <CopyDir>
- <CopyFile>
- <CSPApplication>
- <Credential>

- <Database>
- <Default>
- <Else>
- <Error>
- <ForEach>
- <GlobalMapping>
- <If>
- <IfDef>
- <IfNotDef>
- <Import>
- <Invoke>
- <LoadPage>
- <Log>
- <Manifest>
- <Namespace>
- <Production>
- <Resource>
- <Role>
- <RoutineMapping>
- <Setting>
- <SystemSetting>
- <User>
- <Var>

<Arg>

Parent tags: <Invoke>, <Error>

Passes an argument into a method called via <Invoke> or <Error>.

- Value—Value of an argument.

```
<Error Status="$$$NamespaceDoesNotExist">
        <Arg Value="${NAMESPACE}"/>
/>
```

<ClassMapping>

Parent tag: <Configuration>

Creates a class mapping from a database to the namespace which contains this <Configuration> item.

- Package—Package to be mapped.

- From—Source database used for mapping.

```
<ClassMapping Package="MYAPP"
     From="MYDB"/>
```

<Compile>

Parent tag: <Namespace>

Compiles the specified class name by calling **%SYSTEM.OBJ.Compile**(*Class*, *Flags*).

- Class—Name of class to be compiled.
- Flags—Compilation flags [ck].
- IgnoreErrors—Continue on error? [0]

```
<Compile
    Class="MyPackage.MyClass"
    Flags="ck"
    IgnoreErrors=0/>
```

<Configuration>

Parent tag: <Namespace>

Child tags: <ClassMapping>, <Database>, <GlobalMapping>, <RoutineMapping>

Required as parent tag of configuration tags within <Namespace>. The closing tag (</Configuration>) activates the mappings for the databases in the namespace and updates the .cpf file.

No properties.

```
<Configuration>
    <Database> . . . />
    <ClassMapping> . . . />
/>
```

<CopyClass>

Parent tag: <Namespace>

Copies or moves the source class definition to the target.

- Src—Source class.
- Target—Target class.
- Replace—Overwrite target class? [0]

```
<CopyClass
    Src="MyPackage.MyClass"
    Target="NewPackage.NewClass"
    Replace="0"/>
```

<CopyDir>

Parent tag: <Manifest>

Copies the source directory to a target.

- Src—Source directory.
- Target—Target directory.

- IgnoreErrors—Continue on error? [0]

```
<CopyDir
    Src="${MGRDIR}"
    Target="F:\MyTargetDir"
    IgnoreErrors="0"/>
```

<CopyFile>

Parent tag: <Manifest>

Copies the source file to a target.

- Src—Source file.
- Target—Target file.
- IgnoreErrors—Continue on error? [0]

```
<CopyFile
    Src="${MGRDIR}\messages.log"
    Target="F:\${INSTANCE}_log"
    IgnoreErrors="0"/>
```

<CSPApplication>

Parent tag: <Namespace>

Defines one or more web applications, as defined within the **Security.Applications** class. (Also see Create and Edit Applications for more details on each of these fields.)

- AuthenticationMethods—Enabled authentication methods. (For supported authentication methods and the corresponding values to provide, see the AutheEnabled property in Security.Applications. For example, commonly used values are 4=`Kerberos`, 32=`password`, and 64=`unauthenticated`.
- AutoCompile—Automatic compilation (in CSP settings)? [1]
- CSPZENEnabled—CSP/ZEN enabled? [1]
- ChangePasswordPage—Path to change password page.
- CookiePath—Session cookie path.
- CustomErrorPage—Path to custom error page.
- DefaultSuperclass—Default superclass.
- DefaultTimeout—Session timeout.
- Description—Description.
- Directory—Path to CSP files.
- EventClass—Event class name.
- Grant—List of roles assigned upon logging in to the system.
- GroupById—Group by ID?
- InboundWebServicesEnabled—Inbound web services enabled? [1]
- IsNamespaceDefault—Default application for the namespace? [0]
- LockCSPName—Lock CSP name? [1]
- LoginClass—Path to login page.
- PackageName—Package name.

- PermittedClasses—Permitted classes.
- Recurse—Recurse (serve subdirectories)? [0]
- Resource—Resource required to access web app.
- ServeFiles—Service files? [1]
  - 0—No
  - 1—Always
  - 2—Always and cached
  - 3—Use CSP security
- ServeFilesTimeout—Time, in seconds, of how long to cache static files.
- TwoFactorEnabled—Two-step authentication enabled? [0]
- Url—Name of the web application.
- UseSessionCookie—Use cookies for the session?

```
<CSPApplication
    Url="/csp/foo/bar"
    Description=""
    Directory="C\InterSystems\IRIS\CSP\Democode1"
    Resource=""
    Grant="%DB_%DEFAULT"
    Recurse="1"
    LoginClass=""
    CookiePath="/csp/demo1"
    AuthenticationMethods="64"/>
```

<Credential>

Parent tag: <Production>

Creates or overrides the access credentials.

- Name—Name of the access credentials.
- Username—User name.
- Password—User password.
- Overwrite—Overwrite if the account already exists.

```
<Credential
    Name="Admin"
    Username="administrator"
    Password="123jUgT540!f3B$#"
    Overwrite="0"/>
```

<Database>

Parent tag: <Configuration>

Defines a database within a namespace.

See Configuring Databases in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide* for information about the database settings controlled by the following properties.

- BlockSize—Block size for database (4096, 8192, 16384, 32768, 65536).
- ClusterMountMode—Mount database as a part of a cluster at startup?
- Collation—Default collation for globals created in the database.

- Create—Create a new database (yes/no/overwrite)? [yes]
- Dir—Database directory.
- Encrypted—Encrypt database?
- EncryptionKeyID—ID of encryption key.
- InitialSize—Initial size of the database, in MB.
- ExpansionSize—Size in MB to expand the database by when required.
- MaximumSize—Maximum size the database can expand to.
- MountAtStartup—Mount when launching the installed instance?
- MountRequired—Require mounting of database at every instance startup?
- Name—Database name.
- PublicPermissions—The Permission value to be assigned to the Resource if it must be created. It is ignored if the Resource already exists. Read-only or read-write.
- Resource—Resource controlling access to the database.
- StreamLocation—Directory in which streams associated with the database are stored.

```
<Database Name="${DBNAME}"
    Dir="${MGRDIR}/${DBNAME}"
    Create="yes"
    Resource="${DBRESOURCE}"
    Blocksize="8192"
    ClusterMountMode="0"
    Collation="5"
    Encrypted="0"
    EncryptionKeyID=
    ExpansionSize="0"
    InitialSize="1"
    MaximumSize="0"
    MountAtStartup="0"
    MountRequired="0"
    StreamLocation=
    PublicPermissions=""/>
<Database Name="MYAPP"
    Dir="${MYAPPDIR}/db"
    Create="no"
    Resource="${MYAPPRESOURCE}"
    Blocksize="8192"
    ClusterMountMode="0"
    Collation="5"
    Encrypted="0"
    EncryptionKeyID=
    ExpansionSize="0"
    InitialSize="1"
    MaximumSize="0"
    MountAtStartup="0"
    MountRequired="0"
    StreamLocation=
    PublicPermissions=""/>
```

<Default>

Parent tag: <Manifest>

Sets the variable value if it is not already set.

- Name—Variable name.
- Value—Variable value.
- Dir—Variable value, if a path to a folder or file

```
<Default Name="blksiz"
    Value="8192" />
```

<Else>

Parent tags: <If>

Executed when the conditional defined by the preceding <If> statement is false.

No properties.

```
<If Condition='#
{##class(%File).Exists(INSTALL_"mgr\iris.key")}
'>
<Else/>
<CopyFile Src="C:\\InterSystems\key_files\iris300.key" Target="${MGRDIR}\iris.key"
IgnoreErrors="0"/>
</If>
```

<Error>

Parent tag: <Manifest>

Child tag: <Arg>

Generates an error.

- Status: Error code.
- Source: Source of the error.

```
<Error Status="$$$NamespaceDoesNotExist" Source=>
    <Arg Value="${NAMESPACE}"/>
</Error>
```

<ForEach>

Parent tag: <Manifest>

Defines values for iterations of the specified index key.

- Index—Variable name.
- Values—List of variable values.

```
<ForEach
    Index="TargetNameSpace"
    Values="%SYS,User">
    <!--Code for each iteration of TargetNameSpace-->
</ForEach>
```

<GlobalMapping>

Parent tag: <Configuration>

Maps a global to the current namespace.

- Global: Global name.
- From: Source database of the global.
- Collation: Global collation [IRIS Standard]

```
<GlobalMapping Global="MyAppData.*"
    From="MYAPP" Collation="30"/>
<GlobalMapping Global="cspRule"
    From="MYAPP"/>
```

<If>

Parent tags: <Manifest>, <Namespace>

Child tag: <Else>

Specifies a conditional action.

- Condition: Conditional statement.

```
<If Condition='$L("${NAMESPACE}")=0'>
    <Error Status="$$$NamespaceDoesNotExist" Source=>
        <Arg Value="${NAMESPACE}"/>
    </Error>
</If>
```

<IfDef>

Parent tags: <Manifest>, <Namespace>

Specifies a conditional action if a variable has been set.

- Var: Variable name.

```
<IfDef Var="DBCreateName">
   <Database Name="${DBNAME}"
       Dir="${MGRDIR}/${DBNAME}"
       Create="yes"
       ...
</IfDef>
```

<IfNotDef>

Parent tags: <Manifest>, <Namespace>

Specifies a conditional action if a variable has not been set.

- Var: Variable name.

<Import>

Parent tag: <Namespace>

Imports files by calling **%SYSTEM.OBJ.ImportDir**(*File*,*Flags*,*Recurse*) or **%SYSTEM.OBJ.Load**(*File*,*Flags*).

- File—File or folder for import.
- Flags—Compilation flags [`ck`].
- IgnoreErrors—Continue on error? [`0`].
- Recurse—Import recursively? [`0`].

<Invoke>

Parent tag: <Namespace>

Child tag: <Arg>

Calls a class method and returns the execution result as the value of a variable.

- Class—Class name.
- Method—Method name.
- CheckStatus—Check the returned status?

- Return—Name of variable to write result to.

```
<Invoke Class="SECURITY.SSLConfigs" Method="GetCertificate" CheckStatus="1" Return="CertContents">
    <Arg Value="iris.cer"/>
</Invoke>
```

<LoadPage>

Parent tag: <Namespace>

Loads a CSP page by calling **%SYSTEM.CSP.LoadPage**(*PageURL*,*Flags*) or **%SYSTEM.CSP.LoadPageDir**(*DirURL*,*Flags*).

- Name—URL of CSP page.
- Dir—URL of directory containing CSP pages.
- Flags—Compilation flags [ck].
- IgnoreErrors—Continue on error? [0].

<Log>

Parent tag: <Manifest>

Writes a message to the log specified by the **setup()** method if the log level specified by the **setup()** method is greater or equal to the Level property provided.

- Level—Log level, from -1 (none) to 3 (verbose).
- Text—Log message (string up to 32,000 characters in length).

See Write to the Manifest Log for more information.

<Manifest>

Parent tag: n/a (Root tag, containing all other tags.)

Child tags: <CopyDir>, <CopyFile>, <Default>, <Else>, <Error>, <ForEach>, <If>, <IfDef>, <IfNotDef>, <Log>, <Namespace>, <Resource>, <Role>, <SystemSetting>, <User>, <Var>

No properties.

```
<Manifest>
    <Namespace ... >
        <Configuration>
            <Database .../>
            <Database .../>
        </Configuration>
    </Namespace>
</Manifest>
```

<Namespace>

Parent tag: <Manifest>

Child tags: <Compile>, <Configuration>, <CopyClass>, <CSPApplication>, <Else>, <If>, <IfDef>, <IfNotDef>, <Import>, <Invoke>, <LoadPage>, <Production>

Defines a namespace.

- Name—Name of the namespace.
- Create—Create a new namespace (yes/no/overwrite)? [yes]
- Code—Database for code.

- Data—Database for data.
- Ensemble—interoperability-enabled namespace? [0]

(Other properties are applicable to Interoperability web applications.)

```
<Namespace Name="${NAMESPACE}"
     Create="yes"
     Code="${NAMESPACE}"
     Data="${NAMESPACE}">
          <Configuration>
               <Database Name="${NAMESPACE}" . . . />
          </Configuration>
</Namespace>
```

<Production>

Parent tag: <Namespace>

Child tags: <Credential>, <Setting>

Defines a production.

- Name—Production name.
- AutoStart—Automatically launch production? [0]

```
<Production Name="${NAMESPACE}"
     AutoStart="1" />
```

<Resource>

Parent tag: <Manifest>

Defines a resource.

- Name—Resource name.
- Description—Resource description.
- Permission—Public permissions.

```
<Resource
    Name="%accounting_user"
    Description="Accounting"
    Permission="RW"/>
```

<Role>

Parent tag: <Manifest>

Defines a role.

- Name—Role name.
- Description—Role description (cannot contain commas).
- Resources—Privileges (resource-privilege pairs) held by the role.
- RolesGranted—Roles granted by the named role.

```
<Role
    Name="%DB_USER"
    Description="Database user"
    Resources="MyResource:RW,MyResource1:RWU"
    RolesGranted= />
```

<RoutineMapping>

Parent tag: <Configuration>

Defines a routine mapping.

- Routines: Routine name.
- Type: Routine type (`MAC`, `INT`, `INC`, `OBJ`, `ALL`).
- From: Source database of the routine.

```
<RoutineMapping Routines="MyRoutine"
    Type="ALL" From="${NAMESPACE}" />
```

<Setting>

Parent tag: <Production>

Configures an item in the production by calling the **Ens.Production.ApplySettings()** method.

- Item—Item name.
- Target—Setting type (Item, Host, Adapter).
- Setting—Setting name.
- Value—Value to configure for setting.

```
<Production Name="Demo.ComplexMap.SemesterProduction">
    <Setting Item="Semester_Data_FileService"
        Target="Item"
        Setting="PoolSize"
        Value="1"/>
    <Setting Item="Semester_Data_FileService"
        Target="Host"
        Setting="ComplexMap"
        Value="Demo.ComplexMap.Semester.SemesterData"/>
    <Setting Item="Semester_Data_FileService"
        Target="Adapter"
        Setting="FilePath"
        Value="C:\Practice\in\"/>
</Production>
```

<SystemSetting>

Parent tag: <Manifest>

Sets the value for a property of any Config class that inherits its **Modify()** method from Config.CommonSingleMethods.

- Name—*Class.property* of the Config class.
- Value—Value to assign to property.

<User>

Parent tag: <Manifest>

Defines a user; if PasswordVar is included, the user's password must be provided in the specified variable name.

- Username—Username.
- PasswordVar—Name of variable containing user password (see <Var> below).
- Roles—List of roles to which user is assigned.
- Fullname—User's full name.

- Namespace—User's startup namespace.
- Routine—User's startup routine.
- ExpirationDate—Date after which user login is disabled.
- ChangePassword—Require user to change password on next login?
- Enabled—Is user enabled?
- Comment—Optional comment.

```
<User
    Username="Clerk1"
    PasswordVar="clerk1pw"
    Roles="Dataentry"
    Fullname="Data Entry Clerk"
    Namespace=
    Routine=
    ExpirationDate=
    ChangePassword=
    Enabled=
    Comment=""/>
```

<Var>

Parent tag: <Manifest>

Defines and sets variables that can be used in the manifest.

- Name—Variable name.
- Value—Value to assign to variable.

```
<Var Name="Namespace" Value="MUSIC"/>
<Var Name="GlobalDatabase" Value="${Namespace}G"/>
<Var Name="RoutineDatabase" Value="${Namespace}R"/>
<Var Name="AppDir" Value="C:\MyApp\${CFGNAME}"/>
<Var Name="GlobalDatabaseDir" Value="${AppDir}\${GlobalDatabase}"/>
<Var Name="RoutineDatabaseDir" Value="${AppDir}\${RoutineDatabase}"/>
<Var Name="Resource" Value="%DB_${Namespace}"/>
<Var Name="Role" Value="${Namespace}"/>
<Var Name="CSPResource" Value="CSP_${Namespace}"/>
```

5.3 Using the Manifest

You can run a manifest manually, or as part of an installation.

5.3.1 Manually

In the %SYS namespace, enter the following command in the Terminal:

```
%SYS>do ##class(MyPackage.MyInstaller).setup()
```

You can pass an array of variables to the **setup()** method by reference. Each subscript is used as a variable name, and the node as the value. For example:

```
%SYS>set vars("SourceDir")="c:\myinstaller"
%SYS>set vars("Updated")="Yes"
%SYS>do ##class(MyPackage.MyInstaller).setup(.vars,3)
```

The second argument in this example (3) is the log level.

5.3.2 During Installation

Export the manifest class as `DefaultInstallerClass.xml` to the same directory where the InterSystems IRIS install (either .msi, setup_irisdb.exe, or irisinstall) is run. It is imported into %SYS and compiled, and the **setup()** method is executed.

Note: Make sure that the user running the installation has write permission to the manifest log file directory. The installation does not proceed if the log file cannot be written.

In addition, if you are installing with a single-file kit, you will need to extract the files from the .exe file into the installation directory to get the expected behavior.

Note that if you use the export technique, you cannot pass arguments directly to the **setup()** method. The following sections describe how to pass arguments on Windows or on UNIX®, Linux, and macOS:

5.3.2.1 On Windows

You can modify the .msi install package to pass variable name/value pairs to the **setup()** method.

You can also use command-line arguments with the installer to pass the location of the exported manifest class, variables, log file name, and log level, as shown in the following example:

```
setup.exe INSTALLERMANIFEST="c:\MyStuff\MyInstaller.xml"
INSTALLERMANIFESTPARAMS="SourceDir=c:\mysourcedir,Updated=Yes"
INSTALLERMANIFESTLOGFILE="installer_log" INSTALLERMANIFESTLOGLEVEL="2"
```

Note: Variable names passed using the `INSTALLERMANIFESTPARAMS` argument may contain only alphabetic and numeric characters (`A-Za-z0-9`) and underscores (_), and may not start with an underscore.

For more information, see the Command Line Reference.

5.3.2.2 On UNIX®, Linux, and macOS

On UNIX®, Linux, and macOS systems, you can set environment variables to define the location of the exported manifest class, variables, log file name, and log level prior to running either **irisinstall** or **irisinstall_silent**, as shown in the following example:

```
ISC_INSTALLER_MANIFEST="/MyStuff/MyInstaller.xml"
ISC_INSTALLER_PARAMETERS="SourceDir=/mysourcedir,Updated=Yes"
ISC_INSTALLER_LOGFILE="installer_log"
ISC_INSTALLER_LOGLEVEL="2"
./irisinstall
```

For more information, see Unattended Installation Parameters.

5.4 Installation Manifest Examples

This section contains examples of installation manifests that do the following:

- Blank Template
- Create a Namespace
- Compile After an Upgrade

5.4.1 Blank Template

The following is a basic manifest template:

Class Definition

```
Include %occInclude

/// Simple example of installation instructions (Manifest)
Class MyInstallerPackage.SimpleManifest
{

XData SimpleManifest [ XMLNamespace = INSTALLER ]
{
<Manifest>
   <Log Level="3" Text="Start manifest" />
   <Namespace Name="">
      <Import File="" />
      <Invoke Class="" Method="" CheckStatus="" Return="">
         <Arg Value=""/>
      </Invoke>
   </Namespace>
   <CopyFile Src="" Target="" IgnoreErrors=""/>
   <CopyDir Src="" Target="" IgnoreErrors=""/>
   <Log Level="3" Text="End manifest" />
</Manifest>
}

/// This is a method generator whose code is generated by XGL.
ClassMethod setup(ByRef pVars, pLogLevel As %Integer = 3,
   pInstaller As %Installer.Installer,
   pLogger As %Installer.AbstractLogger)
   As %Status [ CodeMode = objectgenerator, Internal ]
{
    #; Let our XGL document generate code for this method.
    Quit ##class(%Installer.Manifest).%Generate(%compiledclass, %code, "SimpleManifest")
}

}
```

5.4.2 Create a Namespace

The manifest in this example creates a namespace (MyNamespace) and three databases. The MyDataDB and MyRoutinesDB databases are the default databases for globals and routines, respectively, while the MyMappingDB database is another globals database. The `<GlobalMapping>` tag creates a mapping to MyMappingDB for `t*` globals in the MyNamespace namespace, which overrides the default mapping.

Class Definition

```
Include %occInclude

/// Simple example of installation instructions (Manifest)
Class MyInstallerPackage.SimpleManifest
{

XData SimpleManifest [ XMLNamespace = INSTALLER ]
{
<Manifest>
    <Namespace Name="MyNamespace" Create="yes"
        Code="MyRoutinesDB" Data="MyDataDB">
      <Configuration>
        <Database Name="MyRoutinesDB" Create="yes"
            Dir="C:\MyInstallerDir\MyRoutinesDB"/>
        <Database Name="MyDataDB" Create="yes"
            Dir="C:\MyInstallerDir\MyDataDB"/>
        <Database Name="MyMappingDB" Create="yes"
            Dir="C:\MyInstallerDir\MyMappingDB"/>
        <GlobalMapping Global="t*" From="MyMappingDB"/>
      </Configuration>
    </Namespace>
</Manifest>
}

/// This is a method generator whose code is generated by XGL.
```

```
ClassMethod setup(ByRef pVars, pLogLevel As %Integer = 3,
   pInstaller As %Installer.Installer,
   pLogger As %Installer.AbstractLogger)
   As %Status [ CodeMode = objectgenerator, Internal ]
{
    #; Let our XGL document generate code for this method.
    Quit ##class(%Installer.Manifest).%Generate(%compiledclass,
    %code, "SimpleManifest")
}

}
```

5.4.3 Compile After an Upgrade

The manifest in this example recompiles code in a namespace called APPTEST, and is intended to be run after an upgrade. It performs the following tasks:

1. Writes a message to the manifest log indicating the start of the manifest installation process.
2. Sets the current namespace to APPTEST, which contains the code that needs to be recompiled after the upgrade.
3. Invokes the method **%SYSTEM.OBJ.CompileAll()** to compile the classes in the namespace.
4. Invokes the method **%Routine.CompileAll()** to compile the custom routines in the namespace.
5. Writes a message to the manifest log indicating the end of the manifest installation process.

Class Definition

```
Include %occInclude

/// Simple example of installation instructions (Manifest)
Class MyInstallerPackage.SimpleManifest
{

XData SimpleManifest [ XMLNamespace = INSTALLER ]
{
<Manifest>
    <Log Level="3" Text="The Installer Manifest is running."/>
    <Namespace Name="APPTEST">
        <Invoke Class="%SYSTEM.OBJ" Method="CompileAll" CheckStatus="1">
            <Arg Value="u"/>
        </Invoke>
        <Invoke Class="%Routine" Method="CompileAll" CheckStatus="1"/>
        </Namespace>
    <Log Level="3" Text="The Installer Manifest has completed."/>
</Manifest>
}

/// This is a method generator whose code is generated by XGL.
ClassMethod setup(ByRef pVars, pLogLevel As %Integer = 3,
   pInstaller As %Installer.Installer,
   pLogger As %Installer.AbstractLogger)
   As %Status [ CodeMode = objectgenerator, Internal ]
{
   #; Let our XGL document generate code for this method.
   Quit ##class(%Installer.Manifest).%Generate(%compiledclass, %code, "SimpleManifest")
}

}
```

6

Configuring Multiple InterSystems IRIS Instances

You can install and simultaneously run multiple instances on a single machine. To do so, install InterSystems IRIS as for a single installation, giving each instance a unique name, a unique installation directory, and a unique port number.

Please refer to the Multiple InterSystems IRIS Instances section of the *System Administration Guide* for more detailed information.

There are additional considerations for configuring access to applications when running multiple instances. For details on this, see Access Applications on Multiple InterSystems IRIS Servers.

Note: The installer automatically configures URLs with root-level application paths to route toward the instance it is currently installing for the convenience of users who only need one InterSystems IRIS instance installed at a time, and who therefore do not need to specify an instance name in the URL.

However, this means that if you have a system with multiple instances configured and you uninstall the most recently configured instance, requests to root-level application paths no longer succeed. You must manually update the Web Gateway application access profiles for the / (root) and /csp paths to direct requests to a default application server other than the one which you have uninstalled. You can then remove the obsolete server access profile for that instance.

7

Changing the InterSystems IRIS Language

When you install InterSystems IRIS, all supported language-specific utility DLLs are installed in the *install-dir*\bin directory. Each DLL contains localized strings and messages.

The format of the name of the DLL is UTILaaa.DLL, where *aaa* is a 3-letter code that signifies the following languages:

Code	Language
CHS	Chinese (Simplified)
DEU	German (Standard)
ENU	English (United States)
ESP	Spanish (Spain)
FRA	French
ITA	Italian (Standard)
JPN	Japanese
KOR	Korean
NLD	Dutch (Standard)
PTB	Portuguese (Brazilian)
RUS	Russian

For information about changing the locale of an InterSystems IRIS installation, see Using the NLS Settings Page of the Management Portal in the "Configuring InterSystems IRIS" chapter of the *InterSystems IRIS System Administration Guide.*

Note: You can change only among 8-bit locales or Unicode locales, not from 8-bit to Unicode or vice versa. See the %SYS.NLS entry in the *InterSystems Class Reference* for more information.

8

Troubleshooting Web Server Setup

If you have installed a web server prior to installing InterSystems IRIS, then the InterSystems IRIS installation process will configure the web server automatically. This section provides some troubleshooting steps for addressing web server setup.

8.1 Documentation Links Not Working

If links to documentation within the Management Portal (or the Windows Launcher) are not working, you may need to redirect the documentation links. There are different steps for Apache and Microsoft IIS web servers.

8.2 Windows and IIS

8.2.1 VS Code Troubleshooting

Additional steps may be required to allow VS Code to connect with your InterSystems IRIS instance. See Configure IIS to Enable VS Code for details.

If VS Code works, but debugging does not, you may need to enable the **WebSockets** feature.

8.2.2 Restarting IIS

You must restart the IIS web server for some changes to take effect.. It is recommended that you restart your web server after making any changes to the web server or Web Gateway configurations, including after the installation process is completed. To restart the IIS web server:

1. Using a Windows command line interface, run the following command to stop the web server:

   ```
   sc stop W3SVC
   ```

2. After the web server stops, run the following command to start the web server again:

   ```
   sc start W3SVC
   ```

8.2.3 Repairing or Modifying Your Instance

You can initiate auto-configuration of the IIS web server after installation in the event it was skipped during the installation process. To do so, run an installation using the same executable file you originally used to install the instance. Choose to **Repair** or **Modify** depending on whether the **CSP for IIS** component was included in your initial installation:

If the **CSP for IIS** component *was* originally installed, but IIS was either not configured or the configuration was manually deleted, you can run a **Repair** to initiate auto-configuration of the IIS web server.

Important: The only way to include the **CSP for IIS** component without auto-configuring your instance is to specify `CSPSKIPIISCONFIG=1`. This flag can only be specified for an unattended installation, or when executing the installation file from the command line. If you did not include this flag during your installation, you should not run a **Repair**. You can instead **Modify** the instance to initiate auto-configuration.

If **CSP for IIS** was *not* included in your original installation, you can **Modify** your installation and select the **CSP for IIS** component. This will cause the installer to initiate auto-configuration of the IIS web server.

In either case, after the installation finishes you should verify additional configuration steps in order for all features to work properly:

1. To enable Management Portal access, set the CSPSystem account credentials for your instance within its server access profile. For details, see Step 6 of Automatic Configuration Behavior.
2. To enable the Launcher links, you need to configure the following InterSystems IRIS Server Manager parameters:
   - Set **Web Server Port** to the port number your web server is using (80 by default).
   - Set **CSP Server Instance** to the lowercase version of your instance name (for example, `iris`).