Sametime 2.0

Sametime Configuration Management

Design Document

Chapter

1

Sametime Configuration Management

Design Document

Scope

The purpose of this document is to improve upon the process of software configuration management for the Sametime product. This includes the building, packaging, and kiting environments and process, the movement of components between development and International groups, and the install and uninstall of the Sametime server and client components.

History

Sametime 1.0

In the Sametime 1.0 product, we had many inter-dependencies between groups. We also did not fully understand these dependencies until late in the development cycle. DataBeam functioned as the integration point between Iris, Ubique, and International. DataBeam would collect the components from the development groups, build the DataBeam components, package and kit one or more language versions, and then distribute this to the various groups. In some cases, this process would take three to four days in order to collect, package together, and redistribute.

File drops received by DataBeam proved to be problematic. The files contained in the these drops were checked into DataBeam's SourceSafe database and in some cases such as install scripts, code had to be merged. The International file drops contained a massive number of files. Dealing with these file drops was extremely difficult to manage and very error prone. We eventually setup very strict procedures where multiple people were involved with the check-in process and would double-check each other.

Sametime 1.5

Because of the issues encountered with Sametime 1.0, two requirements were envisioned. Maintain independence between the development groups and divide the product into components.

Early in the 1.5 development cycle, these two requirements became the paradigm of "separate" installs, where each development group would independently develop the install of their respective component. Then these components would be grouped under a "Master" install that would call each of the component installs. This process was used only for the server installation. The client install continued using the same process that was

used in the Sametime 1.0 product. New issues arose from this new process. These issues will be discussed in the following section "Deficiencies with Sametime 1.5".

One goal that was accomplished was the ability for everyone who needed to check in files in to source control was given the ability to perform that task. This greatly reduced the responsibility upon a single person to verify that the thousands of files that compose Sametime were correctly checked into source control. The only exception to this was Ubique. They currently do no have direct access to the Sametime source control database. Therefore, they had to provide their code updates using the existing 1.0 file drop process.

Deficiencies with Sametime 1.5

Install and Uninstall

The Sametime server installation presents a less than ideal install experience for the user. The installation has the appearance of being separate installs chained together. This presents several problems for the user. For instance, the progress bar can become lost behind other windows if the user iconifies the installation window or changes the windows' stacking order by using Alt-Tab. A critical dialog box for one install hides behind a window of another install. This causes the installation to appear to hang. A user then has to Alt-Tab to find the dialog and respond in order for the installation to continue. In addition, the server install can take a long time. This leaves the user at a loss for what is happening during the install. There are several "flashing" screens that appear which break up the flow of the install.

Debugging individual installation components is generally not a problem. However, most installation problems occur when a product installation is done from the master installation. This is caused by the interaction of the installation of the server components. Debugging needs to be started from the master install, but it is not feasible with InstallShield to debug across multiple installation processes. Therefore, when errors occurred during a full Sametime installation, it consumed much time to debug. For a single install, InstallShield's interactive debugger is an excellent tool for finding install issues. Because this cannot be used for Sametime, a divide and conquer method of adding print statements to the installation script must be employed.

There was an initial install specification, but as the product cycle progressed, the specification was not updated to reflect changes and additions to the process. Therefore, each development group has no overall knowledge of the install and uninstall process. Consequently, the integration has been error prone has cost time trying to debug the installation.

Because the components that comprise Sametime are "separate" components, this means that there is a significant amount of redundant work for each component. Failure of any development group to do this work results in errors when integrating the components into Sametime.

The Sametime server install relies heavily on initialization files to gather information about the install that all components then use during the various phases of the install. This requires unnecessary synchronization between the separate components. In addition, the location of the main initialization file has presented an issue with the CD image. It had to be moved to the system directory of the target machine.

Code Drops

For subsequent code drops only the deltas, (changes) are needed, but instead, code drops contain the entire component project, including old content. Even with release notes, this has proven to be quite error-prone when integrating updates into the Sametime product.

There have been several instances where "bad" code drops have been delivered. When these drops have been placed into the build, this causes the build to be bad and significant amounts of time are wasted.

SourceSafe

SourceSafe has proven to be inadequate for Sametime when trying to branch, label and merge. It is also easily corruptible. This has posed the greatest risk with the international group checking in files to the DataBeam SourceSafe. Even using PC Anywhere, there have been instances where international files checked into the DataBeam Sametime database somehow became corrupted in the process.

In addition, no formal training has been provided to the different development groups, so the information that DataBeam has gathered regarding the "dos and don'ts" of SourceSafe has not been filtered throughout all the Sametime development groups.

International

It was assumed early on that localization would occur at the various International group locations. This assumption was made without input from the international group. This assumption was also incorrect in that each development team is responsible for creating their component's install to include international support. This support includes using an external resource DLL that contains all the install string resources instead of using Installshield's value.shl.

Component Dependencies

Because of the underlying product dependencies, the components required numerous interdependencies, i.e. the MeetingServer APIs were all written to accept Notes calls, and vice versa. Also there are templates, (Iris), that contain applets, (DataBeam). By strictly segregating components between development groups, this creates a circular component interdependency.

Chapter 2

Process Refinement

Overview

The main objective of refining the process is to provide a simplified process that takes components produced by various development groups and produce the required installable images. Because the development groups are scattered throughout the world across various time zones, it is very important to maintain a certain level of independence between groups. Elimination of roundtrips of components between groups is essential to success. If one development group has to send a component to another group and then wait for the modified component to come back before continuing, days can be added to the overall process.

Another objective is to enable the various development groups to package the final product. Primarily, this allows each group to test their components in an integrated environment with components released by other groups. In addition, the International group will be able to package and test localized versions of the product. Although each group will have the capability to produce the gold version, it is probably best to have one group responsible for the gold version. For the localized versions, it may be best for the International group to take on the responsibility of producing the localized gold versions.

Terminology

Component. A component for this purpose of this document is a major element of Sametime. Generally, this refers to the Domino DNA, Community Services, and Meeting Services. Each component may also contain sub-components.

Development Group. A development group is an organization that provides a major component of the product. The current development groups are Iris, Ubique, DataBeam, and International.

Building. Building is the process of compiling and linking source code to produce binaries. This could also encompasses the inclusion of pre-built binaries from other projects or 3rd party packages.

Packaging. Packaging is the process of creating an InstallShield image from the binaries.

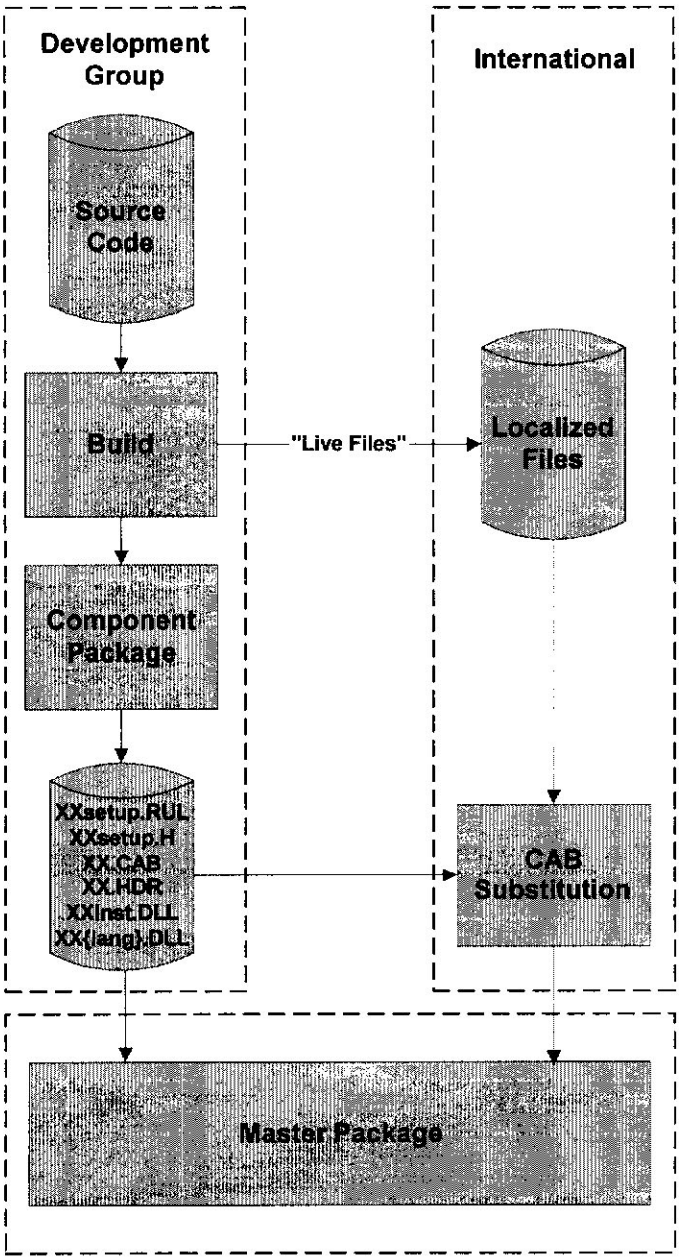
Kiting. Kitting is the process of creating a localized version of a package.

Live and Dead Files. The concept of live and dead files is used by International. *Live files* are those files that need to be localized and *dead files* do not need to be localized.

Master Package. The Master Package is the final installable InstallShield image for the product. Some refer to this as the integrated build.

Implementation

The following diagram shows the overall new process:



This diagram shows the flow of components between groups as well as within groups. However, the scope of this document is to focus on the integration points between groups. The three major areas are comprised of building, kitting, and packaging.

The area designated as *Development Group* constitutes Iris, Ubique, and DataBeam. Each development group is responsible for their own source code control environment and build process for creating the binaries. Each development group will also perform a component package. From the build process, each development group must transfer the *live files* to the International group. The advantage here is that the International group does not have to deal with any files except *live files*. Any file received by International is assumed to be localizable. The burden is on the development group to identify those files that need to be localized which is something that had to be done anyway.

After the binaries are built, the development group then creates a component package. This component package consists of the following files:

- XXsetup.RUL
- XXsetup.H
- XX.CAB
- XX.HDR
- XXinst.DLL
- XX{lang}.DLL

where XX is a component designator such as UB for Ubique's Virtual Places, DB for DataBeam's Meeting Center, and IR for Iris' Domino.

The XXsetup.RUL and XXsetup.H files contain common published entry points for use by the master installation in order to install the major component. The XX.CAB file contains all of the binaries of the major component. The XX.HDR file contains information about the XX.CAB file. The XXinst.DLL contains any special code needed by the major component such as dialog boxes for sub-components, etc. The XX{lang}.DLL file contains all strings used within the installation. The International group will localize this file.

These files generated by the component packaging are available to all other development groups. In this way, all groups can build and test their components with the rest of the product. The International group will have a script file that will perform a CAB substitution. This script file will replace localizable files in the CAB with the correct localized version for that language.

Each development group would have the ability to create the master package. InstallShield, using the component package generated by each group, creates the master package. The only difference for International is the prior running of the CAB substitution script. If a development group needed to package their components separately from the rest of Sametime, that group would create its own master packaging setup.

NOTE: It is the intent of this process to minimize circular dependencies. Files should typically flow in a single direction. However, localized files are currently included into all versions of Sametime (North America, Western Europe, etc.). Concerning a version such as North America, the server contains both English and French. In addition, the Meeting Center components such as the Java applets and the AppShare Host contain localized files for all languages, not just English and French for North America. This area warrants further discussion. One thought is to have the development groups produce something like a US English version that contains no localized files. The International group could then

inject the localized files per the new process to create versions such as North America, Western Europe, etc.

Points of Integration

The process outlined above has the significant advantage of the components flowing in only one direction. In other words, no group has to be dependent upon another group to make modifications to their components to keep the process moving forward. This process also spreads general responsibilities around to each group. For instance, DataBeam is responsible for the signing of their templates. International is responsible for the signing of their templates. Reducing these types of dependencies will greatly increase a particular group's productivity.

There are two major integration points described above that warrant discussion between the groups. First, what is the best mechanism for the development groups to transfer the *live files* to the International group? Second, what is the best mechanism for the development groups to share their component packages with the other groups.

To date, we have tried several approaches including FTP or Domino document libraries for making code drops and SourceSafe. Each method has its advantages and disadvantages. One approach may work well for the *live files*, but perhaps not so well for the component packages.

Live Files

Currently, we have a SourceSafe database located at DataBeam. This database contains both *live files* and *dead files* of which we only have to maintain *live files* moving forward. DataBeam accesses this database via a local area network. Other groups such as International access this through PCAnywhere. Direct SourceSafe client access is unreliable and too slow for other groups across a wide area network. If we continue with the SourceSafe approach, it would probably be best to locate this SourceSafe database within International. Other groups would access this via PCAnywhere.

The following outlines several alternative approaches:

3) FTP. Each group would FTP their files to a specific location. Each group would be permitted to modify only their files. Other groups would have read-only access. The advantage is that all groups can use FTP. The disadvantage is that there is no versioning information maintained. It is also possible that two individuals within the same group upload the same file at the same time causing corruption problems. To circumvent corruption, strict procedures have to implemented and followed by each group. Another alternative is to use a mirrored FTP. Lotus currently uses this today. The advantage is that each group would have local high-speed access to the FTP directory. The replication of the directory happens later.

4) Document Library. Each group and possibly multiple locations within a group such as International would have a local copy of a document library. Everyone would share this document library. Updates to the document library are distributed via replication. The advantage to a document library is that each group would have local access to it. Replication would occur later. In addition, ACLs could be implemented to make sure

that each group only has read-only access to another group's files to prevent accidental modifications. In addition, ACLs could be applied at a person level as well.

5) Mail-in Database. The advantage to a mail-in database is that everyone has access to e-mail. The disadvantages are that only the group owner of the database would see the updates unless the database was replicated or made available to other groups. In the case of the *live files*, this may be irrelevant because only International is the only recipient of these files. The other disadvantage of a mail-in database is the size of the files. Many e-mail systems do not handle the transfer of large files.

Component Package

Delivering the component package files to other groups is very similar to delivering *live files* to the International Group. Except that in this case, all groups need access to all other groups' deliverables.

The FTP and Document Library methods outlined above would probably work best. The only downside to the document library approach is the replication schedule. It may take some time for one group to receive another group's files. The FTP method would be almost instantaneous. These methods need further exploration and probably some variation will prove to be the best method.

Integration Standards

To ensure that all the development groups provide components that seamlessly integrate into a single product, each development group will implement the following standards.

1) Each development group will prefix all files in the component package with the following **Development Group Prefix**,

   - DataBeam **DB**
   - Iris **IR**
   - Ubique **UB**

   As an example, the component package delivered by DataBeam would look like,

   DBdata1.CAB

   DBdata1.HDR

   DB{language}.DLL

   DBresource.H

   DBsetup.H

   DBsetup.RUL

2) The current **language set** for Sametime will be the following:

   Brazilian Portuguese

Danish

Dutch

English

Finnish

French

German

Italian

Japanese

Korean

Norwegian

Simplified Chinese

Spanish

Swedish

Thai

Traditional Chinese

3) InstallShield component calls in InstallShield context usually rely on the global variable, "**MEDIA**". By default, InstallShield initializes the value of this variable to "**DATA**". The value of this variable may be changed to suit the nature of the component function that is being called, but the value must remain consistent throughout the duration of the install. The value of this variable will differ based upon whether it is in a stand alone or integrated environment. To facilitate this, any function that uses any component function calls will accept this media variable as an input parameter. An example of the function definition, (defined in the component setup header file), would be the following,

prototype DBSetupForLanguage (STRING);

In a stand alone environment, a call to this function would be,

DBSetupForLanguage (MEDIA);

In an integrated environment a call to this function would be,

DBSetupForLanguage (<media>);

Where media is a global variable, (defined in the "master" setup.rul), and is defined as follows,

- DataBeam **DataBeam_Media**
- Iris **Iris_Media**
- Ubique **Ubique_Media**

Chapter

3

Server Installation

Overview

The Sametime server install will consist of a single install (Master Package) that has been integrated to contain the files (components) defined in Chapter 2, Overview. This single install will present the user with the option of installing the Meeting Center and/or the Community Server. Domino DNA, as the underlying platform will always be installed unless Domino is already present on the server. These components and their responsible development groups are defined as follows,

- **Sametime integrator.** Creates and maintains the Master Package that integrates the component API calls. These APIs will be defined in the section, ComponentAPI. The Sametime integrator will be one of the development groups.
- **DataBeam.** Creates and maintains the Meeting Center component.
- **Iris.** Creates and maintains the Domino DNA component.
- **Ubique.** Creates and maintains the Community Server component.

This paradigm of integration allows for all the development groups to retain complete component independence, and simultaneously solves most, if not all, Sametime1.5 integration issues by integrating into a single install. It also facilitates the idea of a componentized install.

On uninstall of the Sametime server, all components that were installed will be removed. The user will be prompted to remove or leave all user data.

Worldwide Geography

For worldwide geographies, the server will be packaged and kited in a single language format. Use of the built in select language InstallShield support will not be used such that multiple languages will be packaged together into a single install. Instead, the CD browser will be responsible for detecting the language selected from user input and subsequently selecting the appropriate "language" install. For installs downloaded from the Internet, there will be a link to each "language" install.

Component API

Because the components that comprise Sametime will be integrated such that the final Master Package is a single install, this means that each development group will have to develop their component install using the paradigm of APIs. This means that each

development group will provide the list of files as defined in Chapter 2, Overview, and a specification documenting the calls that the Master Package should make into each component install script.

Each component install should provide the functionality to either install standalone or integrated into a larger product. To achieve this, each component install will check for a define called "**INTEGRATED_PRODUCT**". This definition exists in the "master" setup.rul file. Each component install will also have "#idefs" based on the INTEGRATED_PRODUCT definition such that if it is defined, then the standalone code will not be compiled in the integrated scenario.

Each component install will provide the following general "API" calls for use in an integrated client product.

- **<Development Group Prefix>SetupForLanguage (<media>);** This function accepts one input parameter, media, and performs any component language specific operations needed for that component. The main operation is to set the **SELECTED_LANGUAGE** InstallShield variable for use when selecting language file components to install. Other operations would be, loading the component's language resource DLL, selecting a translated license agreement, etc.

- **<Development Group Prefix>ProcessBeforeDataMove (<media>);** This function accepts one input parameter, media, and performs any component specific operations needed before the component's file data is moved to the target system. Operations performed here would be, setting a file component based upon user input in the ShowDialogs () function, deleting old, unused files in the case of an upgrade install, etc.

- **<Development Group Prefix>ProcessAfterDataMove (<media>);** This function accepts one input parameter, media, and performs any component specific operations needed after the component's file data is moved to the target system. Operations performed here would be, creating a desktop icon on the target system,

Sample Code

TBD...

Chapter

4

Client Installation

Overview

The Sametime client install will consist of a single install (Master Package) that has been integrated to contain the files (components) defined in Chapter 2, Overview. This single install will present the user with the option of installing the AppShare client component and/or the Connect client component. These components and their responsible development groups are defined as follows,

- **Sametime integrator.** Creates and maintains the Master Package that integrates the component API calls. These APIs will be defined in the section, ComponentAPI. The Sametime integrator will be one of the development groups.
- **DataBeam.** Creates and maintains the AppShare Host component.
- **Ubique.** Creates and maintains the Community Server component.

On uninstall of the Sametime server, all components that were installed will be removed. The user will be prompted to remove or leave all user data.

Worldwide Geography

For worldwide geographies, the client will be packaged and kited in two versions, multiple language and single language selections. Use is made of InstallShield's built in language selection support for multiple language packages. The client packages will look like the following:

Korean	Korean
Japanese	Japanese
Nordics	Danish, Dutch, English, Finnish, French, Norwegian, Swedish
North America	French, English
Simplified Chinese	Simplified Chinese (PRC)
Thai	Thai
Traditional Chinese	Traditional Chinese (Taiwan)

Western Europe	Brazilian Portuguese, English, French, German, Italian, Spanish

Component API

Because the components that comprise Sametime will be integrated such that the final Master Package is a single install, this means that each development group will have to develop their component install using the paradigm of APIs. This means that each development group will provide the list of files as defined in Chapter 2, Overview, and a specification documenting the calls that the Master Package should make into each component install script.

Each component install should provide the functionality to either install standalone or integrated into a larger product. To achieve this, each component install will check for a define called "**INTEGRATED_PRODUCT**". This definition exists in the "master" setup.rul file. Each component install will also have "#idefs" based on the INTEGRATED_PRODUCT definition such that if it is defined, then the standalone code will not be compiled in the integrated scenario.

Each component install will provide the following general "API" calls for use in an integrated client product.

- **<Development Group Prefix>SetupForLanguage (<media>);** This function accepts one input parameter, media, and performs any component language specific operations needed for that component. The main operation is to set the **SELECTED_LANGUAGE** InstallShield variable for use when selecting language file components to install. Other operations would be, loading the component's language resource DLL, selecting a translated license agreement, etc.

- **<Development Group Prefix>ProcessBeforeDataMove (<media>);** This function accepts one input parameter, media, and performs any component specific operations needed before the component's file data is moved to the target system. Operations performed here would be, setting a file component based upon user input in the ShowDialogs () function, deleting old, unused files in the case of an upgrade install, etc.

- **<Development Group Prefix>ProcessAfterDataMove (<media>);** This function accepts one input parameter, media, and performs any component specific operations needed after the component's file data is moved to the target system. Operations performed here would be, creating a desktop icon on the target system,

Component Specific API

In addition, there are some component specific API calls. These calls are defined as AppShare and Connect function APIs.

AppShare Component

- **DBCheckForNetscape (<media>);** This function accepts one input parameter, media, and if a Netscape browser is installed on the target system. If a Netscape browser is detected, then this function will also determine the number of Netscape

browsers installed, and the version of each one. The number of browsers detected will be stored in the global variable, **DB_Number_Netscape_Browsers**. This function also selects the AppShare Netscape Plugins File Component to be selected if a Netscape browser is detected on the target system.

- **DBShowDialogs ();** This function checks the global variable, DB_Number_Netscape_Browsers. If this number is greater than 1, the function displays a dialog prompting the user to select which version of Netscape to reference when installing the AppShare Host component.

Connect Component

- **VPDialogShowAskConnectServer ();** This function checks for an existing install of the Connect client on the target system. If found, this function then reads the Community Server name from the existing **connect.ini**. If not found, then the user is prompted via a dialog box for the Community Server name.